0262

NUWC-NPT TM 922098

LIBRARY USE ONLY

NAVAL UNDERSEA WARFARE CENTER DIVISION
NEWPORT, RHODE ISLAND

Technical Memorandum

DEVELOPMENT OF A FLOW VISUALIZATION TECHNIQUE FOR TRANSIENT FLUID FLOW

Date: 31 December 1992

Prepared by: Kenneth M. LaPointe
Launcher Systems
Development Division
Launcher and Missile
Systems Department

TECHNICAL LIBRARY
NAVAL UNDERSEA WARFARE CENTER
NEWPORT DIVISION
NEWPORT, R.I. 02841-5047

Approved for public release; distribution is unlimited.

LIBRARY USE ONLY

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
31 DEC 1992	**Technical Memorandum**	**31-12-1992 to 31-12-1992**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Development of a Flow Visualization Technique for Transient Fluid Flow	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Kenneth LaPointe**	5d. PROJECT NUMBER **A70322**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Undersea Warfare Center Division,Newport,RI,02841**	8. PERFORMING ORGANIZATION REPORT NUMBER **TM 922098**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Naval Undersea Warfare Center's Independent Research Program**	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
NUWC2015

14. ABSTRACT

This document describes a quantitative flow visualization technique for transient fluid flow. The flow visualization technique uses particle image velocimetry to obtain instantaneous global measurements of the fluid flow field. The particles in the flow field are illuminated with a multiple pulsed laser sheet, and the image of the illuminated particles is then captured on film. The film is digitized on a high resolution scanner, and the digital data are sent to a computer system where either an autocorrelation code or a two-dimensional fast Fourier transform code is sequentially applied to small regions of the digitized picture to compute the average displacement of the particles in each region. Knowing the time between laser pulses allows the velocity vector for each region to be calculated from the displacement data.

15. SUBJECT TERMS
transient fluid flow; flow visualization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**118**	

ABSTRACT

This document describes a quantitative flow visualization technique for transient fluid flow. The flow visualization technique uses particle image velocimetry to obtain instantaneous global measurements of the fluid flow field. The particles in the flow field are illuminated with a multiple pulsed laser sheet, and the image of the illuminated particles is then captured on film. The film is digitized on a high resolution scanner, and the digital data are sent to a computer system where either an autocorrelation code or a two-dimensional fast Fourier transform code is sequentially applied to small regions of the digitized picture to compute the average displacement of the particles in each region. Knowing the time between laser pulses allows the velocity vector for each region to be calculated from the displacement data.

ADMINISTRATIVE INFORMATION

This memorandum was prepared under FY91/92 Project No. A70322, "Development of a Flow Visualization Technique for Transient Fluid Flows," principal investigator K. M. LaPointe (Code 8322), sponsored by the Naval Undersea Warfare Center's Independent Research Program.

The author of this memorandum is located at the Naval Undersea Warfare Center Division, Newport, RI 02841-1708.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ccp	Cubic centimeters of particles
ccw	Cubic centimeters of water
FFT	Fast Fourier transform
f/sec	Frames per second
g/cc	Grams per cubic centimeter
LDV	Laser Doppler velocimetry
PIV	Particle image velocimetry
p/sec	Pulses per second
SG	Specific gravity
l/mm	Liters per millimeter
IR	Infrared
TTL	Transmitter transmitter logic
CW	Continuous wave
WARP	Windows for actuators and rotary stages program
TCP/IP	Transmission control protocol/Internet protocol
SGI	Silicon Graphics Incorporated
HPGL	Hewlett Packard graphics language
Nd:YAG	Neodymium yttrium aluminum garnet

INTRODUCTION

Past experimental methods of examining fluid flow were limited to point measurement techniques such as laser Doppler velocimetry (LDV), hot wire anemometry, and pitot probes. These techniques were accurate measurement tools but were not useful in measuring global flow fields. Other methods, such as dye injection and yarn tufts, helped visualize the global flow field but did not provide quantitative data on the flow itself. A tool for the quantitative measurement of the global flow field needed to be developed for hydrodynamic investigators. The tool would have to be useful for steady state and transient flows.

This document describes a flow visualization technique for transient fluid flow. Particle image velocimetry (PIV) is used to obtain instantaneous global measurements of the fluid flow field. The particles in the flow field are illuminated with a multiple pulsed laser sheet, and the image of the illuminated particles is then captured on film. The film is digitized on a high resolution scanner, and the digital data are sent to a computer system where either an autocorrelation code or a two-dimensional (2D) fast Fourier transform (FFT) code is sequentially applied to small regions of the digitized picture to compute the average displacement of the particles in each region. Knowing the time between laser pulses allows the velocity vector for each region to be calculated from the displacement data.

This document details each phase of the PIV process, from the selection of particles to the display of the computed vectors.

PARTICLE SELECTION

The particles in the flow field are critical to the entire process. The particles should be approximately neutrally buoyant, from 0.95 to 1.10 grams per cubic cm, and between 10 and 40 μ in diameter (Katz and Huang, reference 1). Large particles will not properly track the flow field and in transient flow fields, large particles will lag the flow changes. If the particle is too small (less than about 10 μ), the film will not be able to record its image. Note that the image of the illuminated particle will be a bit larger than the actual particle diameter due to blooming from the reflected laser light. Fluorescent or reflective particles can be used to enhance the visibility of the particles. Fluorescent particles can be designed to fluoresce at a different wavelength than the laser light. This will allow use of a color filter to mask any unwanted reflections and stray laser light. Particle reflectance can be enhanced by a metal coating or a high index of refraction change between the particle and the water.

This project examined many particles of various sizes and shapes, table 1. The 3M large glass bubbles worked well, but the small bubbles were too small for the 3 x 4 inch area of interest. The glass bubbles had to be washed first to remove the silica

coating. The solid polystyrene particles had poor reflectance and were hard to pick up on film. The 3 μ fluorescent particles were bright but too small; they looked more like a dye. The silver coated particles had the best image brightness; however, at 2.6 g/cc they would be poor at following the flow. Fluorescent particles of 30 μ diameter were ordered but did not arrive in time for evaluation. At this time the best particles to use are the 35 μ glass bubbles.

Table 1. Particles Listing

Particle Type	Material	Company	Mean Diameter (μ)	SG (g/cc)
Glass Bubbles	Silicon Glass	3M	8.00	1.10
Glass Bubbles	Silicon Glass	3M	35.00	0.60
Polymer Microspheres	Polystyrene DVB	Duke Scientific	0.5 - 24.0	1.05
Fluorescent Microspheres	Dyed Polystyrene	Duke Scientific	3	1.05
Metallic Coated	Silver Coated Glass	TSI	12	2.6
Plastic Particles	Dyed plastic	Dr. J. Katz	35	1.0

SEEDING DENSITY

The seeding density required for a particular test is dependent on size of the window used for each velocity vector. The window size is a function of the grid density of velocity vectors per picture. Katz and Huang (reference 1) and Adrian (reference 2) have shown that both the FFT and the correlation methods need from 6 to 10 particle pairs per window to yield an accurate velocity vector. Given the grid size and the thickness of the light sheet, the required particle density can be computed. It can be seen that for a given vector density, as the viewing area gets smaller, the particle density increases dramatically. For small viewing areas a large number of particles will cloud the water and obscure the light sheet. A compromise must be made between seeding density and water clarity.

Example:

Window Size:	0.12 cm x 0.12 cm
Experimental Field of View:	10.8 cm x 7.2 cm, (4.25" x 2.83")
Vector Grid:	90 x 60
Thickness of Light Sheet:	0.2 cm
Window Volume:	0.12 cm x 0.12 cm x 0.2 cm = 0.00288 ccw
Number of Particles:	8 particles per window volume.

Particle distribution = particles per window / water volume of window

Particle distribution: 2777 particles/ccw.

Particle volume, 35 μ: $\frac{3.14 \times D^3}{6} = \frac{3.14 \times 0.0035^3}{6} = 2.243 \times 10^{-8}$ ccp.

Volume of particles = Particle distribution in water x Particle volume

Volume of particles: 6.228×10^{-5} ccp/ccw.

Density of particles: 1.05 g/ccp

Particle-Water Weight Ratio = Volume of particles in water x Density of particles

Particle-Water Weight Ratio: 6.540×10^{-5} g/ccw.

Amount of water in experiment: 56.775 m^3 (15,000 gal).

Estimated amount of particles required for experiment :

6.540×10^{-5} g/ccw x 56.775 m^3 = 3.7 kg of particles (8.2 lb).

This example gives a good estimate for the amount of particles required. The actual amount will vary depending on settling, water clarity and light sheet design of the experiment.

CAMERA REQUIREMENTS

To get a respectable viewing area at high resolution, a 35 mm or larger film format must be used. The 35 mm format has the advantage over other formats in cost and availability. A standard 35 mm camera can be used for single frame test shots. A good quality 35 mm camera with a motorized film drive will give from 3 to 6 frames per second (f/sec). A 35 mm Hultcher camera will give a 25 or 40 f/sec speed. A 16 mm LOCAM high speed camera will give high frame rates, but its viewing area will be extremely small. Also, availability of film in the 16 mm format is somewhat limited.

Camera shutter speed settings are a bit confusing. The setting is not the time required for the shutter to open or close, nor is it the time it takes the shutter to traverse the film. It is the total amount of time for which any one section of the film is exposed to light. For all standard 35 mm cameras, exposure time is accomplished by a traveling shutter opening that passes a slit of light across the film. The slit may travel in the horizontal or vertical direction. The width and speed of the traveling slit determine the exposure time. The width of the slit and speed of the slit vary from camera to camera. For the purpose of the flow visualization project, the problem is that the entire piece of film is not exposed at once when using the camera speed settings. One side of the film is exposed before the other.

For the flow visualization work, we want the picture of the entire flow field at the same instant in time. That means that the picture needs to be taken with the shutter in the fully opened position. With the shutter fully opened the laser system can be pulsed to expose the entire frame of the film. To do this a strobe synchronization signal must be sent to the laser when the camera shutter is fully open. On most cameras this is the "X" or flash setting. A flash synchronization setting on a 35 mm camera provides a closed circuit for the flash when the shutter is fully open. By connecting a battery to this circuit a high level pulse can be provided for other electronic setups.

The Hultcher camera used in the experiment had no shutter synchronization and had to be modified to give a pulse before the shutter opens (figure 1). A slotted disk was mounted on the shutter shaft. The disk has a slit that is detected by an infrared (IR) emitter detector set. When the slit passes the detector it will put out a 5 V transmitter transmitter logic (TTL) signal. Since the slit is very small it will be a 5 V TTL pulse. An electronics diagram for the emitter and detector pair is presented in figure 2. The time between the pulse disk output and the camera shutter at the fully open position must be set to allow ample time for the laser system to get ready before the camera shutter is at the fully open position, roughly 10 msec. The pulse delay time can be checked by sending the laser beam, at low power, through the camera shutter while the camera is operating. The laser pulse was adjusted to coincide with the shutter at the fully open position. The pulse delay time was adjusted as needed by turning the pulse disk a few degrees at a time.

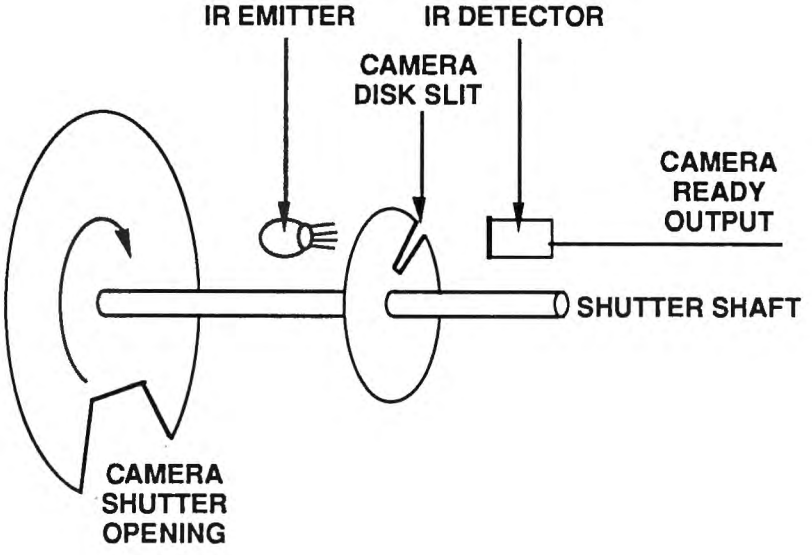

Figure 1. Camera Synchronization Controller

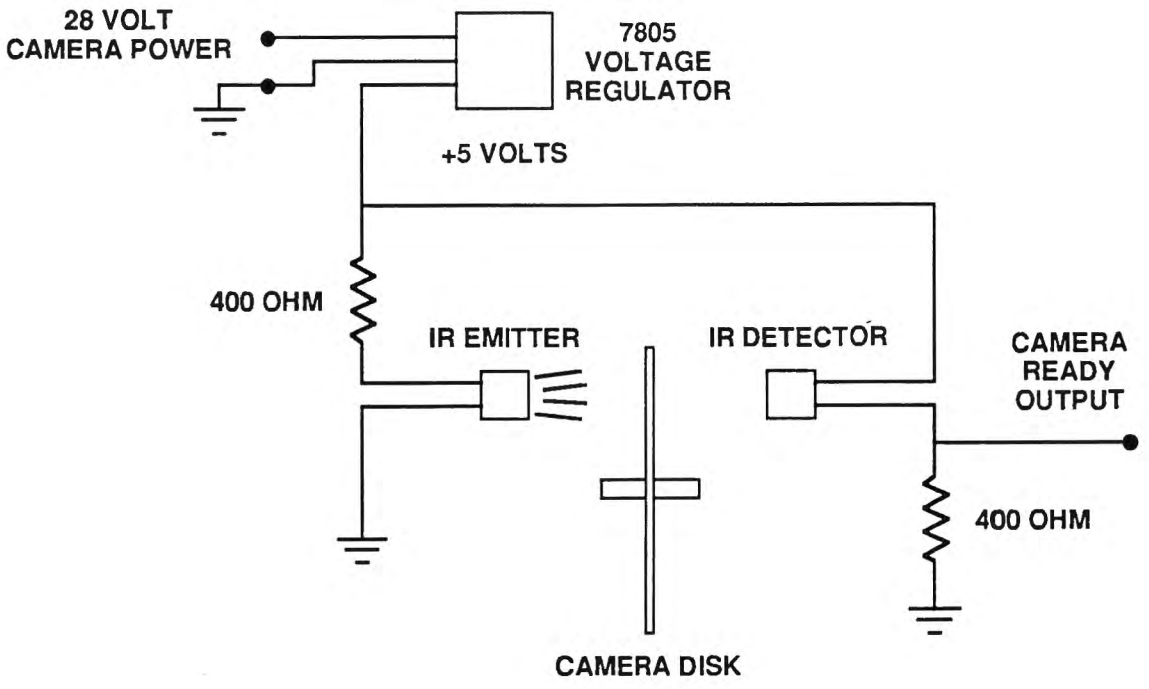

Figure 2. Camera Synchronization Electronics

For most tests, the camera has to be kept out of the flow field. To get the required image magnification a zoom lens was used. The camera was kept as close as possible to the image area. If the camera is too close for the zoom lens alone, a lens and bellows setup can be used. A bellows unit moves the image plane away from the film plane and allows a normal zoom lens to be used as a macro zoom lens. One must be careful of the extremely small depth of field when using the zoom lens and bellows arrangement.

FILM REQUIREMENTS

The viewing area for the flow visualization also depends upon the film resolution. Since color films typically have a low ASA rating and have resolution lower than 80 lines per mm (l/mm) black and white films are the best choice. Table 2 is a partial listing of ASA rating and resolving powers for various films (references 3 and 4). While a 25 W laser may seem powerful, the amount of light going to the film is quite small. The laser pulse is about 3 mJ at 8 kHz, but each pulse is only 20 nsec in duration. So the light exposure to the film is really quite small. Hence, an extremely fast film is required. Since we want very small particles to properly track the flow field, we need high resolution film to capture the particles accurately. Tri-X Pan with an ASA rating of 400 was tested and was found to be too slow. Based on this test and the capabilities of the film, KODAK T-MAX P3200 black and white film was tested. This film has an ASA rating of 3200 and a resolution of 125 l/mm. The film worked well with the particles tested.

Table 2. Film Listing

Film Type	ASA Rating	Resolving Power (l/mm)
BLACK AND WHITE		
Technical Pan 2415	25	320
Plus-X Pan	125	125
Tri_X Pan (TX)	400	100
T-MAX P3200	800 - 3200	125
COLOR FILMS		
Ectar 1000	1000	80
Gold Extapress	1600	40 - 80

Due to the resolution of the film, the size of the particle must be taken into account when setting up the camera field of view. The particle should be a minimum of 3 to 4 lines wide on the negative. It is not easy to calculate the maximum field of view for a given particle. While the particle has a given size, its image on film can be larger due to the reflective properties of the particle, clarity of the water, and laser power. Testing for each particle would have to be conducted to get its image size on film. In the experimental setup used for this project (2 mm light sheet with the laser at 25 W and camera lens setting of f/5.6), the 12 μ silver coated particle produced a 3 line picture with a field of view of 10.8 cm x 72 cm (4.25 inch x 2.83 inch). For the same setup the 35 μ glass bubbles produced a 4 line picture of the particle.

One other particle parameter is the brightness of the image. The particle needs to produce a high contrast image in order for the computer to "see" the particle.

LASER SELECTION

The PIV technique requires a high power, high speed, pulsed laser source. Laser options are limited to a continuous wave (CW) laser with a chopping wheel, (Nd:Yag) or a metal vapor laser. The selected laser must be able to output short pulses, 10 nsec, at high repetition rates, 2 to 10 kHz. The CW laser with a chopper wheel is limited to 4 kHz with a relatively large pulse width of 125 microseconds. The Nd:Yag laser runs around 20 to 200 pps with a pulse width of 7 nsec with per pulse power of around 120 mJ. While the Nd:Yag provides a high power pulse, its use requires two lasers to give double pulsed capability. For this project the best compromise between high repetition rate and power per pulse was a 25 W copper vapor laser. This laser uses copper metal vaporized at extremely high temperatures

and voltages. The copper vapor lases and emits a green yellow beam. The 25 W CJ laser used in this testing puts out 25 W average power at a nominal 8 kHz with a peak power of 3.5 mJ at 8 kHz and 2 mJ at 2 kHz. The laser pulse rate range is from 2 to 10 kHz.

To keep the copper metal vaporized and lasing, the laser tube must be kept hot. To keep the laser hot, it must be running near its design frequency, in this case 8 kHz. For the PIV work laser pulse frequency may be as low as 2 kHz. However, due to the starting and stopping required by the camera, 20 or 40 Hz, the laser off time would allow the system to cool down. A laser interface box, ESC-2000, which will run the laser at 8 kHz normally and will keep the laser running at the required 8 kHz, stop it, run it at the requested frequency and then start it again, is connected to the laser. A laser start pulse will tell the laser to stop running at 8 kHz, open the laser shutter, EMS-2000, and run at the external trigger frequency. This takes about 5 msec. A stop pulse command will tell the laser to stop running at the external trigger frequency, close the shutter and run at the 8 kHz again. This takes about 5 msec.

LIGHT SHEET DESIGN

PIV works by taking a picture of particles in the flow field. The selection of which particles to view is made by forming the laser light into a light sheet and projecting the light sheet into the water (figure 3). The camera is held perpendicular to the light sheet for viewing the flow field. The light sheet must be thin in order to reduce errors caused by particles going at angles to the investigated path.

Due to the large output beam diameter (20 mm) of the copper vapor laser, the laser beam must be focused to a smaller size (about 2.5 mm). The small beam is then passed through a cylindrical lens to form a light sheet. All of the lenses should be set to minimize reflections back into the unstable resonator. Reflections into the unstable resonator will contaminate the quality of the generated light.

For this testing the beam was focused by passing it through a 2 inch f/1000.0 mm lens. An f/125.0 mm lens was then placed after the focusing point to refocus and collimate the smaller beam. The small beam was then passed through the cylindrical lens to form the light sheet.

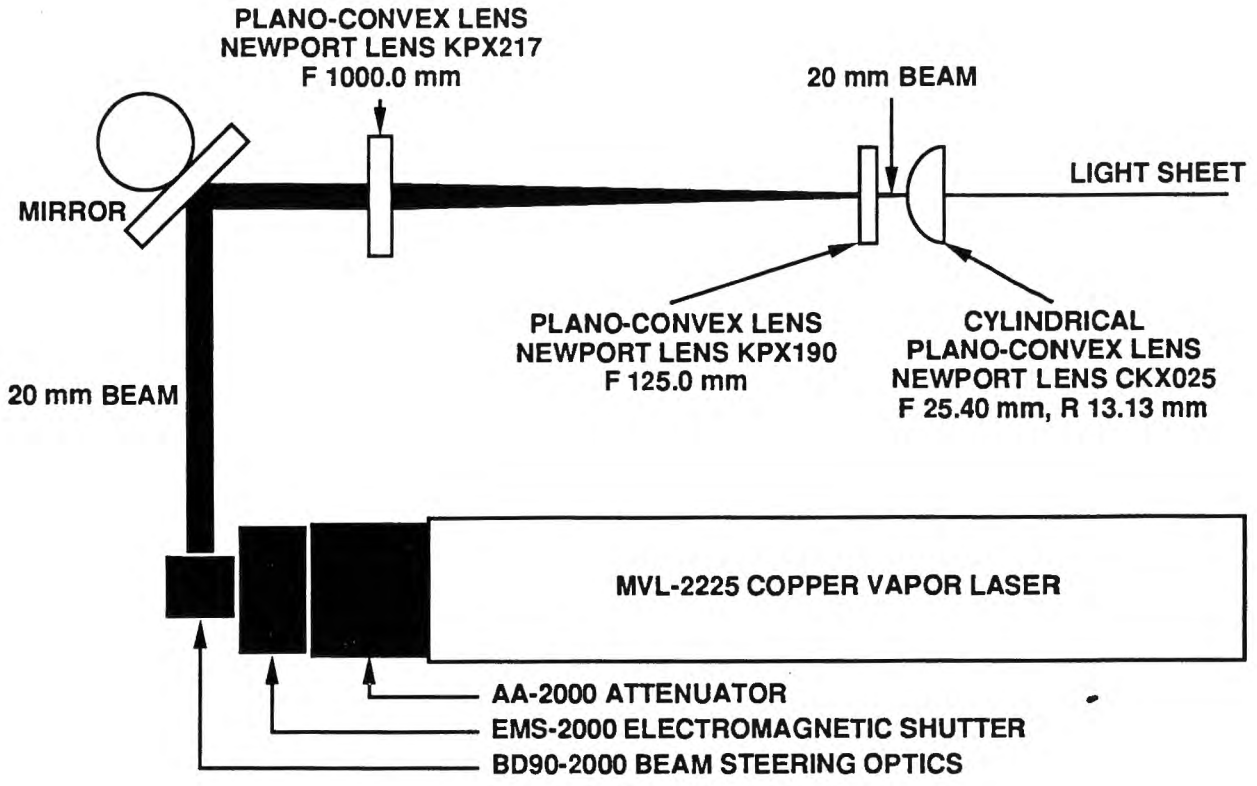

Figure 3. Optical Train

LASER PULSE CONTROL

The laser pulse must be controlled so that the laser will pulse when the camera is at the fully opened position. The camera sends out a ready pulse for the computer to send a signal to pulse the laser. A Masscomp computer was used to control the pulsing of the laser. The computer front panel clocks (figure 4) and the program **laser_control.c** are used to control the laser start and stop pulses and the external trigger pulses. The computer program sets up the delay time between the camera ready synchronization and the laser start synchronization. The program must be told the proper time settings for the shutter delay line, PIV frequency, and number of pulses. The code is currently set to run the laser at one frequency. By using the transient subroutine in the code, the clock can vary the laser frequency during the test. To eliminate false triggering of the laser due to electronic noise, the program sets the trigger level on the Masscomp clock to 2.5 V. This eliminates noise problems but does cause some problems with other Masscomp users. Once the program is finished, the clocks are reset to the standard 1 V setting. The program also puts out a graphic file that displays the clock pulse setup and timing. An example of the timing chart is shown in figure 5.

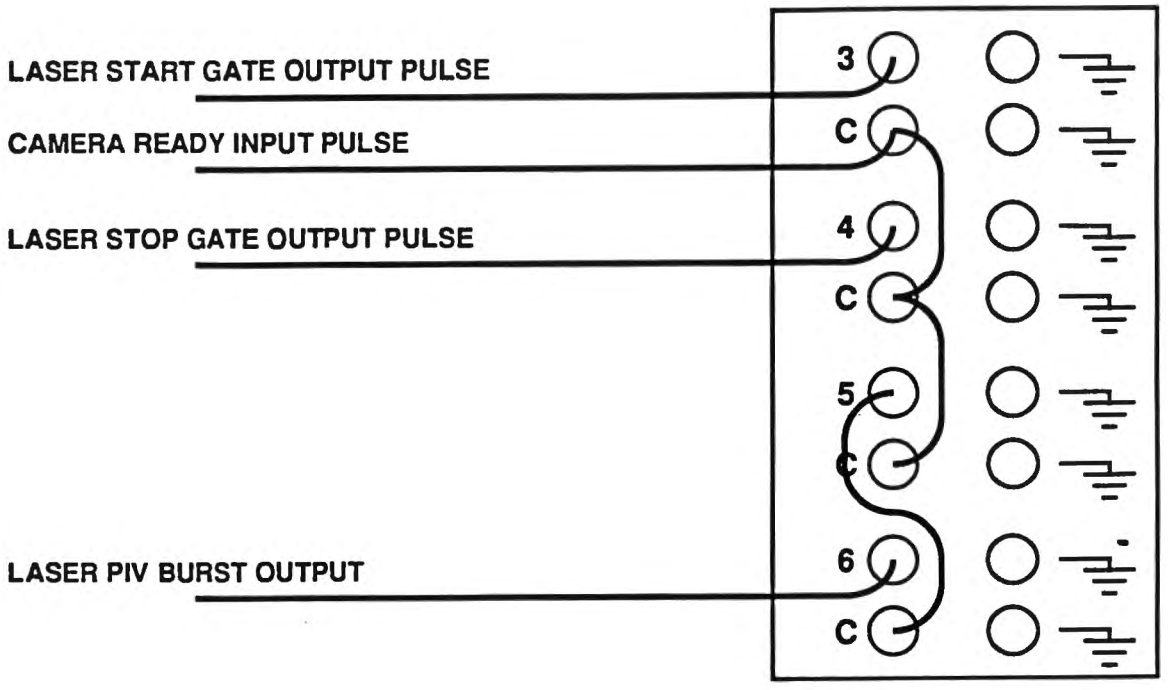

Figure 4. Masscomp CK10 Front Panel

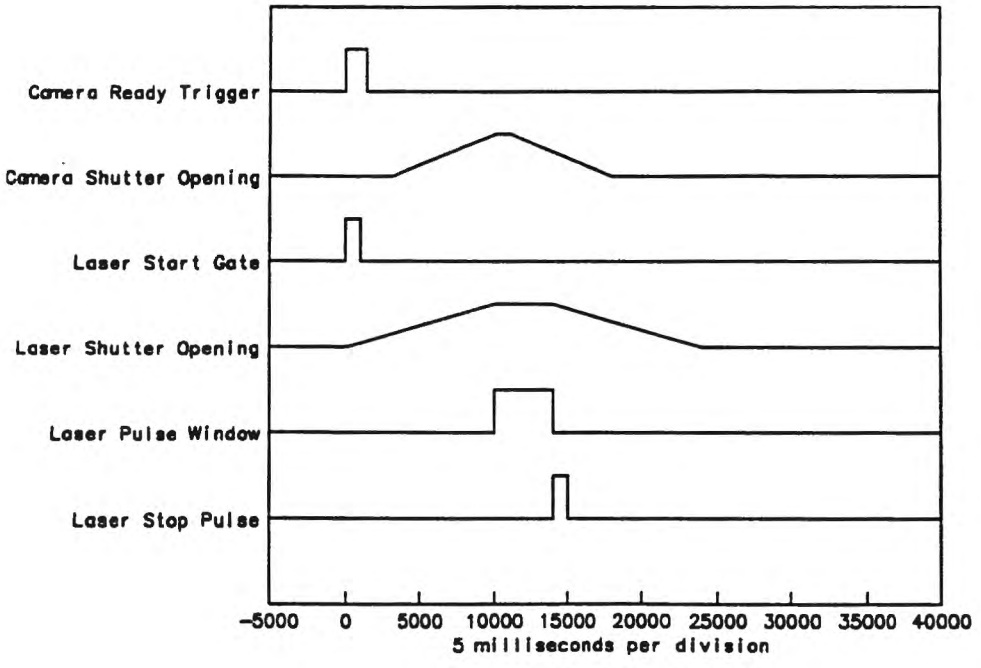

Figure 5. Masscomp Clock Timing Chart

CLOCK SYNCHRONIZATION OPERATIONS

The camera ready pulse will come from the camera ready line and go into the CK3, 4 and 5 C inputs of the CK10. The camera ready pulse is prior to the camera shutter fully open position and is set to account for the camera disk offset, allow time for the laser to stop pulsing at 8 kHz, and open the laser shutter, about 5 msec. The CK3 output, start laser pulse, will tell the laser to stop lasing, open the shutter, and pulse at the external trigger frequency. The start laser pulse is delayed to allow time for the laser shutter to open and at the same time it must be set to be coincident with the camera shutter fully open position. This is done by trial and error. The CK4 output is set to send a close laser pulse once the laser has finished its pulsing. The CK5 output will tell the computer when to pulse and for how long; it is connected into the CK6 C input. The CK 6 output is connected to the laser external trigger input. The external trigger frequency is determined by the flow velocity and viewing area of the experiment.

AUTOMATIC SCANNING SYSTEM

An automatic film scanning system was designed around a Nikon scanner. The system is controlled by software on the PC, which controls the scanning, processing, and film movement (figure 6). The scanning is performed on a Nikon Ls-3510 AF high resolution scanner. The film is moved by two Newport Model 850 linear actuators and one Model 495 rotary actuator. The film movement is controlled by the PC via a Newport PMC-300 Power/Interface Module.

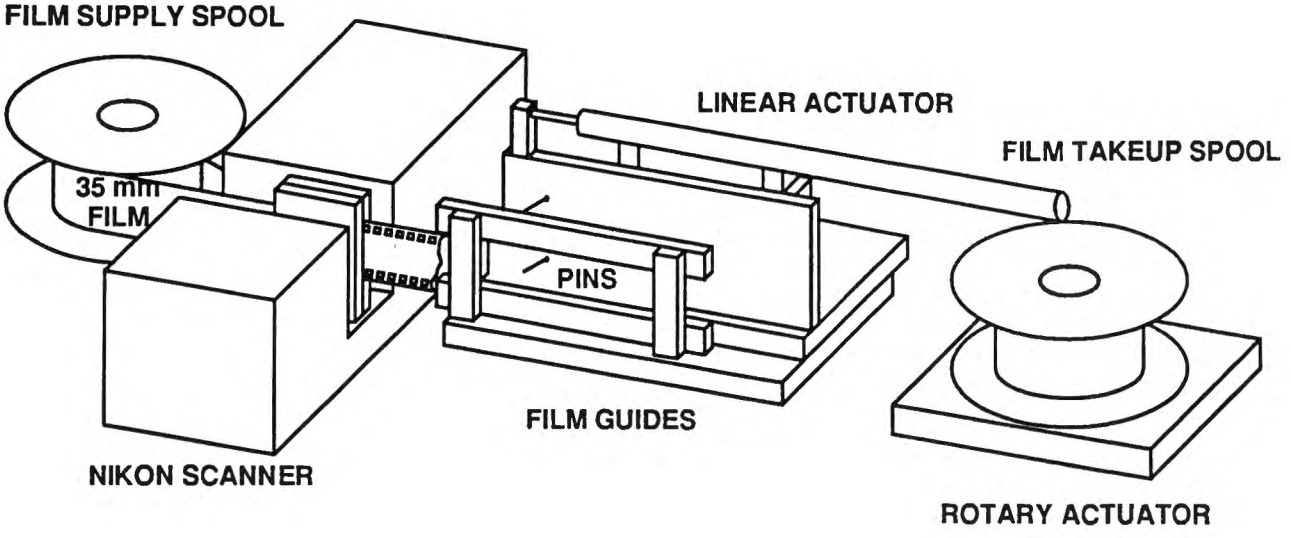

Figure 6. Film Transport System

Prior to running the auto scan software, one must align the film on the film holder and the film transport pins must be aligned with the film sprocket holes. The film transport alignment is accomplished by using the Window for Actuators and Rotary stages Program (WARP) code provided with the PMC-300 control system. It must also be verified that the data analysis software is compiled and running on the Cray and that the Cray is on-line.

CONTROL CODE

The **runpiv.c** program controls the digitizing and submission process by calling other computer codes. The **runpiv.c** code will ask for a run id, starting frame, and number of frames. It will use this information to sequence through the film and create appropriate data files. The files will use the format of **idrun_numberfrframe_number.file_type**, where file type is **.job** for the Cray job file, **.snd** for the Cray transfer shell file, **.put** for the PC file transfer file, and **.dsp** for the computed displacement data file.

FILM SCANNING

The control code calls the **scanpic.c** program to scan the image from the film. The image is scanned on a NIKON LS-3510AF color scanner with a 4500 x 3000 pixel resolution. The image is scanned in as 256 gray shades. Control of the scanner is by the IEEE-488 bus from the PC. The resulting file is stored with a **.img** suffix. The image file is rather large (27 Mb).

JOB SUBMISSION

The control code next runs **submit.bat** to transmit the image and the analysis job to the Cray. The transmission is made using Ethernet TCP/IP protocol. The user must have a current account on the Cray computer and have the ftp network files set up for this account. Transfer of the image file to the Cray can take as long as 1 hour. Files are automatically removed from the Cray after the job is completed. After data analysis is completed the displacement data file is sent to the Silicon Graphics Incorporated workstation for post-processing and display.

FILM ADVANCE

The control code then calls **advance.c** to move the film transport to advance the film to the next frame. The transport comprises two linear actuators mounted to engage and move the film with two pins (figure 6). A rotary actuator is also

mounted to wind the film during the process. The transport pins are moved into the holes on the film and then the film is pulled one frame length. The pins are pulled away from the film and the transport is moved back to the starting position.

PARTICLE IMAGE VELOCIMETRY

Particle image velocimetry uses multiple exposures of moving particles to measure particle velocity. By using a pulsed laser to illuminate the particle at different times and recording the particle images, one can measure the distance between the particles. Knowing the time between pulses allows the velocity to be computed. To visualize this, consider a laser pulse at time t_0. Images of the particles are recorded on film. The laser is pulsed again some time later, time t_1; the particles have moved a distance d, which depends on the particle velocity v and the time between pulses dt. The image of the particles at time t_1 will be shifted from the image of the particles at time t_0. Both exposures are recorded on one piece of film (figure 7).

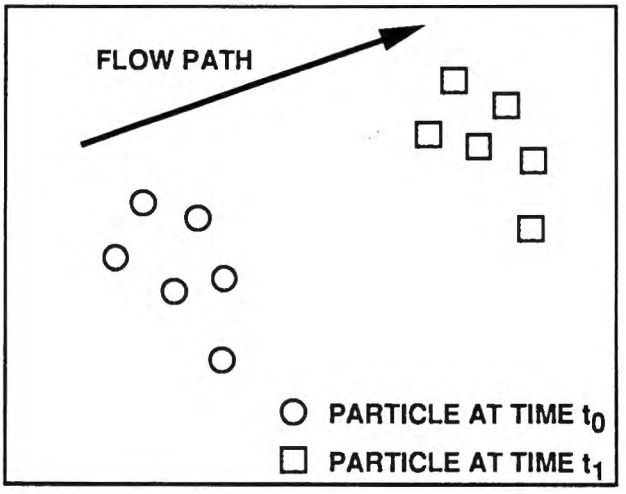

Figure 7. Double Exposed Particle Image

The particle velocity can be computed directly from the picture. A large picture with many particles is broken down into small windows. Each window must be sized so that all of the pulsed images from a particle fit in that window. At large flow speeds this can require large windows. The laser frequency can be increased to reduce the window size, but this does increase the minimum measurable particle velocity. The average velocity of each window is computed and displayed by the computer.

There are two primary methods for computing the average velocity of the particles. The first method is a spatial autocorrelation method that uses overlaying maps of the image and shifts the maps to determine the best correlation. The sec-

ond method uses two 2D FFT's to compute the spatial frequency of the particle pairs.

The image is broken into small windows to compute the average particle velocity for each window. The size of the window must be larger than the maximum expected particle displacement. For high accuracy and wide range the window should be as large as possible. Unfortunately, large windows will lower the number of velocity vectors and grid resolution.

AUTOCORRELATION METHOD

The autocorrelation method uses image shifting to find the best correlation. This is accomplished by taking a copy of the double exposed image and placing that copy directly over the original image (Katz and Huang, reference 1). The initial position of the copy is at the 0,0 point. Now consider that the image is in digital format. For simplicity let the dark pixels be 1 and the clear points be 0 (the gray shade of the digitized image is used in the computer program). Go through each pixel of the original image, take its value, and multiply it by the value of the pixel on the copy directly above it. Do this for all of the pixels and add the values together. This will give the autocorrelation value for the zero position. This will be the peak autocorrelation value, since it has the best correlation (figure 8). If the copy on the top is moved by a single pixel in both the x and y directions, and the process is repeated for each movement, the new value will be the autocorrelation value for that pixel displacement. The peak value from the zero position is compared with the next highest peak. The next highest peak will be for the position when the t_0 particles on the copy best overlay the t_1 particles on the original. The distance between the zero position and the next highest peak is the average displacement for those particles. This is the basic process behind autocorrelation PIV.

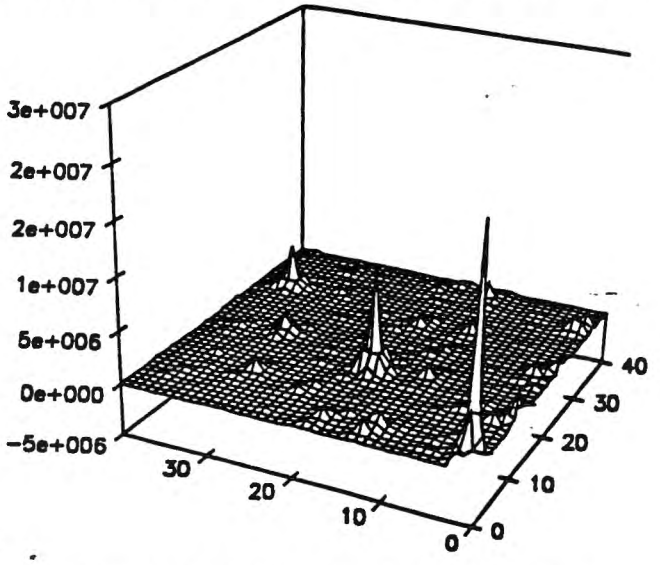

Figure 8. Autocorrelation Peaks

The **autocore.c** program was written to take a multiple pulsed exposure of the particle image and perform the autocorrelation. The program reads in the digitized picture and breaks up the picture into windows. The average displacement for each window is computed. Each window should be sized to have about 8 particle pairs in it for the autocorrelation to be accurate. A smaller window will run faster and give more vectors, but the accuracy of the vector will be poorer since the number of particle pairs will be lower. A large window will be more accurate but will take very long to compute and will yield few velocity vectors. The size of the window will have to be determined by looking at the negative. The program also has to know how big, in pixels, the particle image appears. This forces the code to go past the zero overlap point, which is the diameter of the particle in pixels.

The particle displacement vector is computed by finding the distance from the zero point to the location of the next highest correlation peak. The location of that peak may not provide an accurate measurement of the average displacement of the particles in the window. To get a better average displacement value, the average location value of the peak and its surrounding area is computed. This is performed in the code by averaging the correlation weighted locations to compute the best average peak location.

The code will go through each window and compute the autocorrelation for the positive x and y, and for the positive x and negative y, and output the vectors into a data file.

This code takes a considerable amount of time (1 to 4 hours) on the Cray to compute all of the vectors. If one had sufficient knowledge of the flow field one could reduce the time of computation by giving the code good initial starting points and limited velocity bands.

TWO-DIMENSIONAL (2D) FAST FOURIER TRANSFORM (FFT) METHOD

The other method to compute particle displacements uses 2D FFT (Adrian, reference 2). Given a window of multiple particle exposures (figure 9), a 2D FFT can be used to get the average spatial displacement between particles. A 2D FFT is performed on a copy of the original image; a display of the magnitude output will show the interference fringes (figure 10), identical to the optically produced Young's fringes . A 2D FFT on the interference fringes will produce a peak at the average displacement point (figures 11 and 12). Both the highest peak and the second highest peak are found. The ratio of the highest to the second highest peak can be used as an indicator of the quality of the data. A ratio of 1.25 or higher has been shown to reflect good data (Keane and Adrian, reference 5).

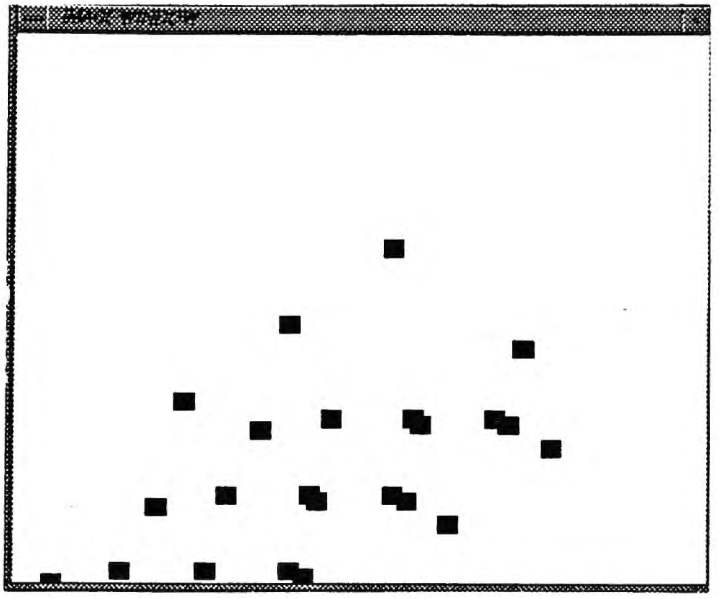

Figure 9. Simulated Multiple Laser Pulsed Particle Field

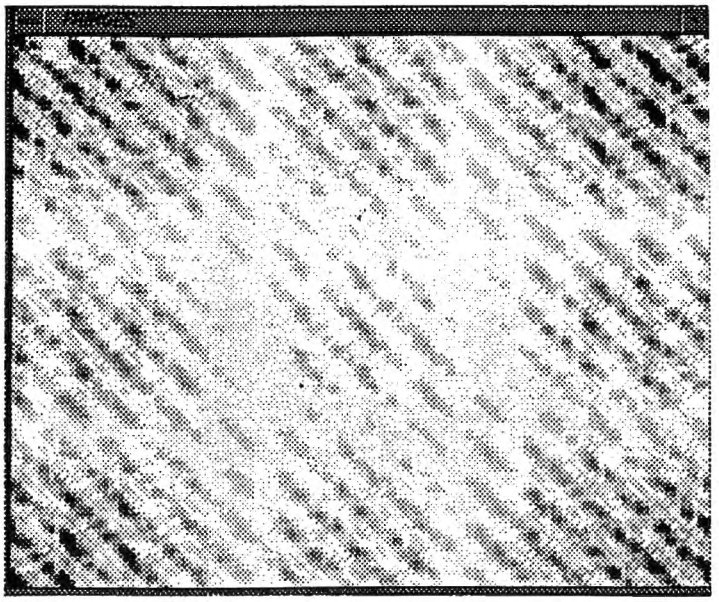

Figure 10. Computed Young's Fringes

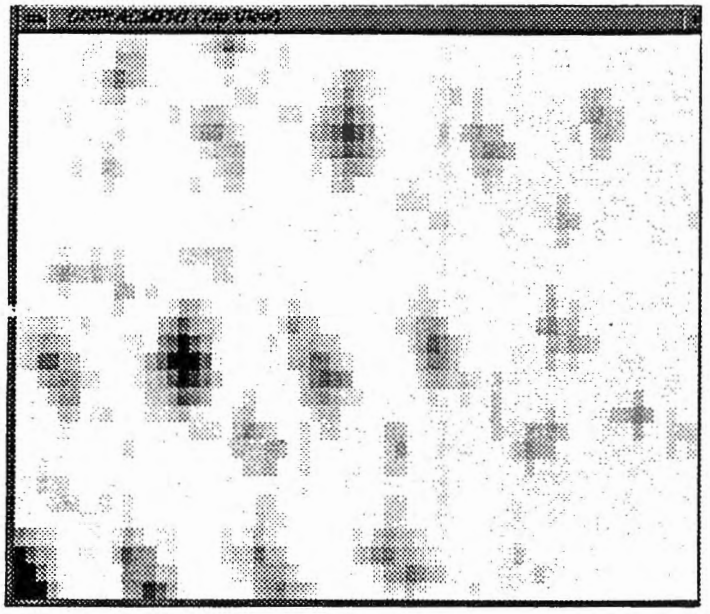

Figure 11. Two-Dimensional Display of Peak Correlation

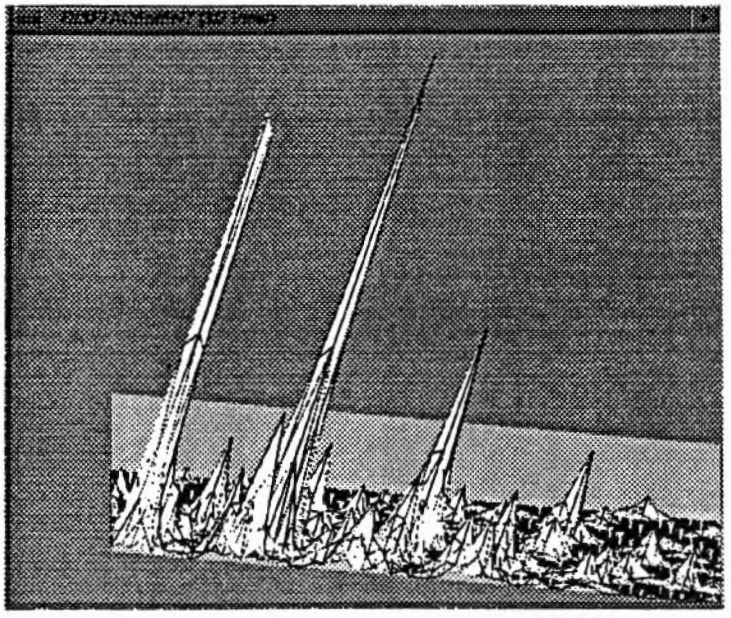

Figure 12. Three-Dimensional Display of Peak Correlation

Limitations do exist for the 2D FFT method. The image window must encompass about 8 particle pairs to be accurate and the order of the FFT must be greater than half the greatest particle displacement, also known as the Nyquist criterion. High order FFTs can be implemented by zero padding of the image window. There are two versions of the code using the 2D FFT method. The first uses a C version of the FORTRAN 2D FFT code in the Numerical Recipes book. This code can be run on the SGI, taking about 2 hours, or on the Cray, taking about 1 hour. Using the Cray intrinsic 2D complex FFT subroutine reduces the computation time to around 5 minutes.

DIRECTIONAL AMBIGUITY

Since PIV is represented as a series of dots on the film there is no information as to actual particle direction. If the direction of the flow is known in one axis, the direction can be determined for the other axis. If no direction is known, as is the case in some flows, the computer will be able to determine angle but not the direction of the flow. Adrian (reference 6) has shown that this ambiguity can be resolved by a technique called image shifting. An additional displacement, larger than the most negative particle displacement, is added to the particles by moving the camera or the film during the filming process. Landreth and Adrian (reference 7) developed a technique that uses Pokel cells for electro-optical movement of the image. The artificial displacement is removed during processing and the direction can be resolved. While this method provides an increased vector window it yields a smaller number of vectors per frame and a smaller field of view. For most flows a high vector density is required and the direction can be determined from other sources.

DATA PRESENTATION

After the displacement vectors are computed they can be displayed on the SGI IRIS. The **vector.c** code will display the vectors on the screen and also dump an HPGL file for plotting the vectors. The vector code can dump all of the vectors or try to eliminate the extraneous vectors. The code uses several methods to eliminate the bad vectors. The first method is to use the ratio of the highest to second highest peak. The ratio can be adjusted to balance the quality of the data with the amount of the data. The code also loops through each computed vector and compares the magnitude and angles of the selected vector to those at the adjacent vectors. If there are not at least two adjacent vectors within a given percentage of the selected vector, the selected vector is not displayed.

EXPERIMENTAL SETUP

A small test chamber was set up to test the PIV system (figure 13). The chamber was kept small to reduce the amount of particles used per test. A 10-gallon glass aquarium tank was used to hold water. A small foot-operated bellows pump pushed water out of a 5/8 inch copper tube. The light sheet was approximately 0.2 cm wide and was aligned with the tube exit. A 35 mm Nikon F2 camera was used with a 200 mm zoom lens on a bellows extension. The camera was set approximately 4 feet from the copper tube. The field of view was 108 cm x 72 cm.

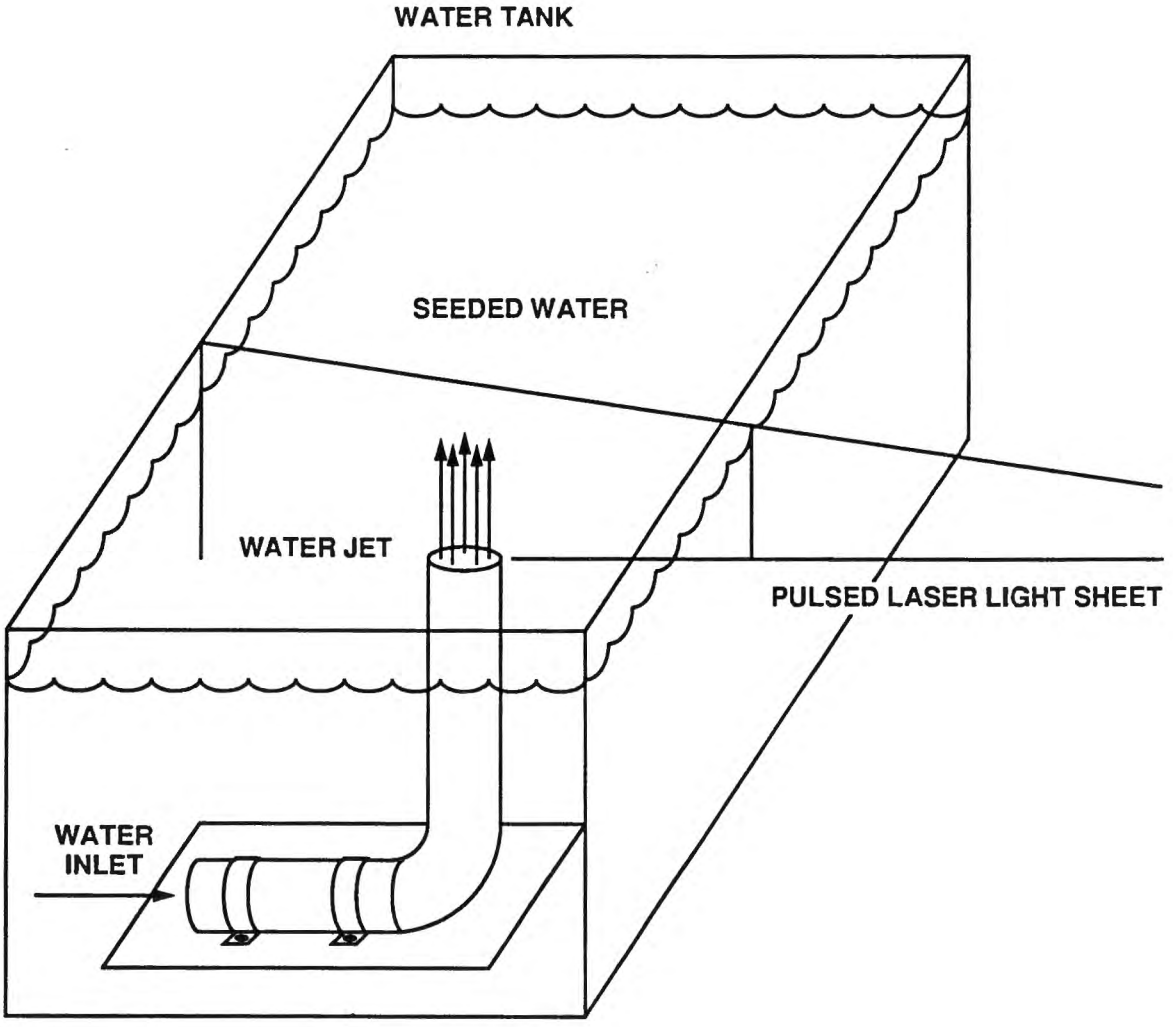

Figure 13. Experimental Setup

The laser was always run at maximum power (25 W). The typical laser frequency was 2 kHz. Various f stop settings were tried. Low settings produced a good image intensity but a poor depth of field; high settings limited the amount of light to the film. The best setting for this setup was f/5.6.

A number of experiments were tried to find the best number of pulses per frame. The minimum required for PIV is 2. Three pulses typically gave better results. When using more than 3 pulses per frame the image tended to loose clarity due to the increased background image noise. A setting of 3 pulses per frame is recommended for PIV.

The Hultcher camera was tested at 25 f/sec. The film was not put through the beater spool so that the film would be constantly moving. This film movement provided image shifting for resolving the directionality of the flow. The window size must be enlarged to fit the increased displacement of the particles. Unfortunately, the larger window size yields fewer vectors. The camera was run in its normal mode to record the jet flow at 25 f/sec. Unfortunately, the film was not developed properly and there was no time left in the program to repeat the test.

EXPERIMENTAL RESULTS

The PIV system provided global velocity information on the flow field. The first test (figure 14) shows a jet of water starting up in a quiescent tank; flow direction is in the positive y direction. All data are displayed with no effort taken to remove the extraneous (bad) vectors. These data were collected using a low seeding density of the 12 μ silver coated particles. The laser was set at 10 kHz producing 25 W. Ten pulses were used with a 2 mm thick light sheet. Figure 15 shows the data after the image has been cleaned up. The water plume and vortices generated at startup are evident in both graphs.

The second test shows a starting flow with 3 pulses at 2 kHz (figures 16 and 17). The startup vortices are evident in figure 18. In figure 16 the indication of vortices is lost due to the removal of the particle diameter limited displacement vectors. The magnification of the image onto the film must be sized to make sure that the small velocities are not lost in the image noise. The third test shows a steady state jet flow, with raw data in figure 18 and reduced data in figure 19.

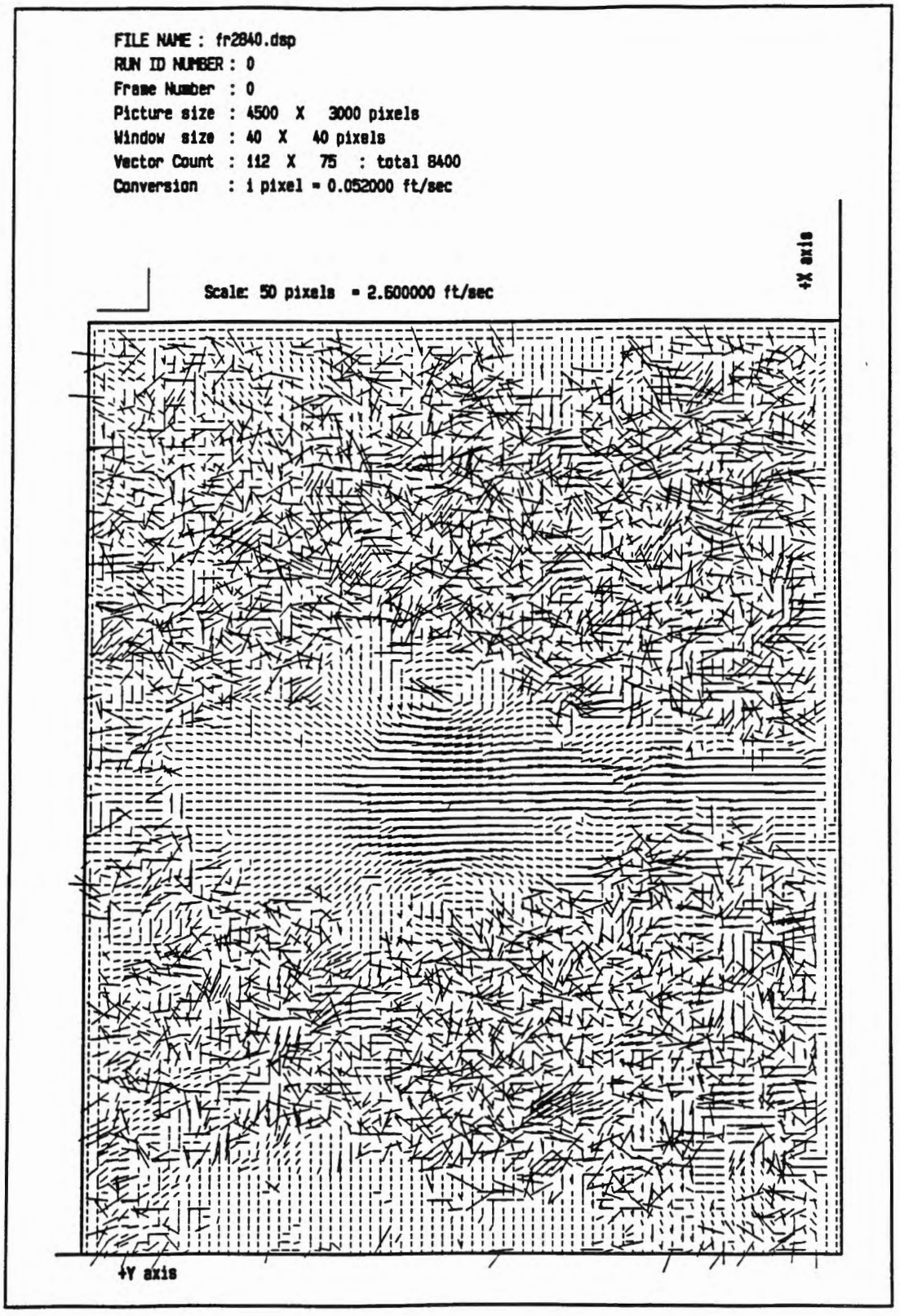

Figure 14. Startup Jet Flow, 10 kHz Laser Pulse (Raw Data)

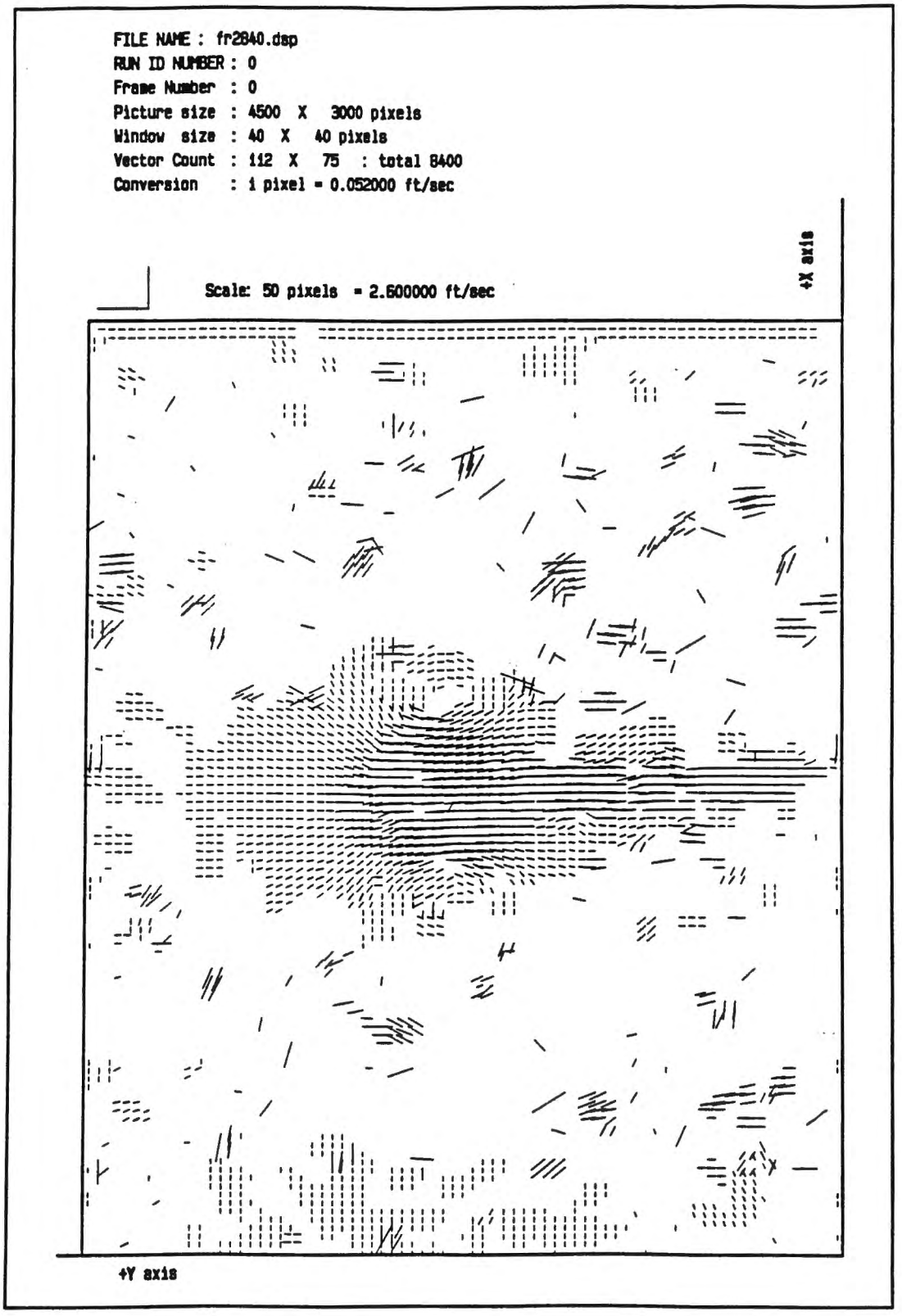

Figure 15. Startup Jet Flow, 10 kHz Laser Pulse (Reduced Data)

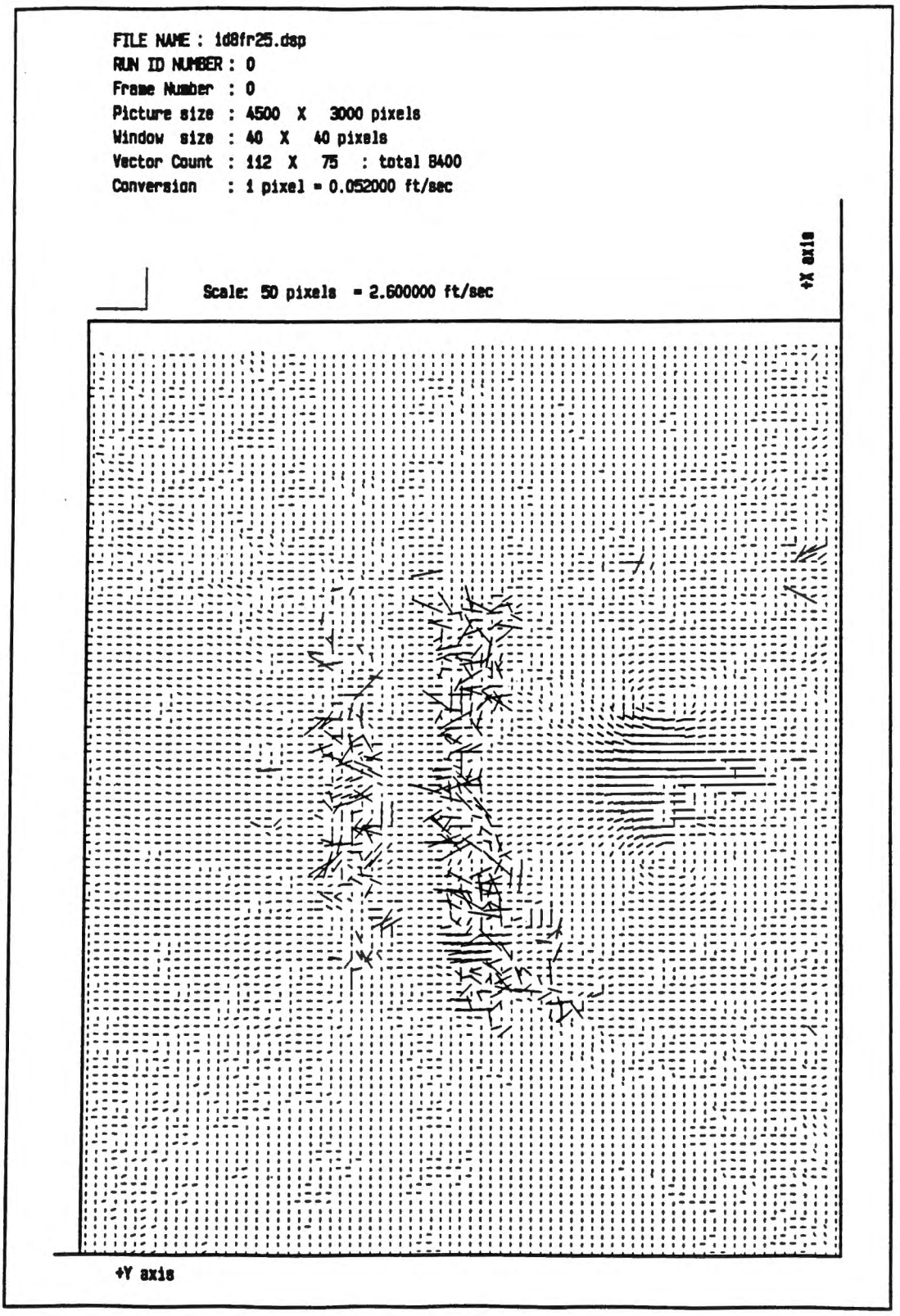

Figure 16. Startup Jet Flow, 2 kHz Laser Pulse (Raw Data)

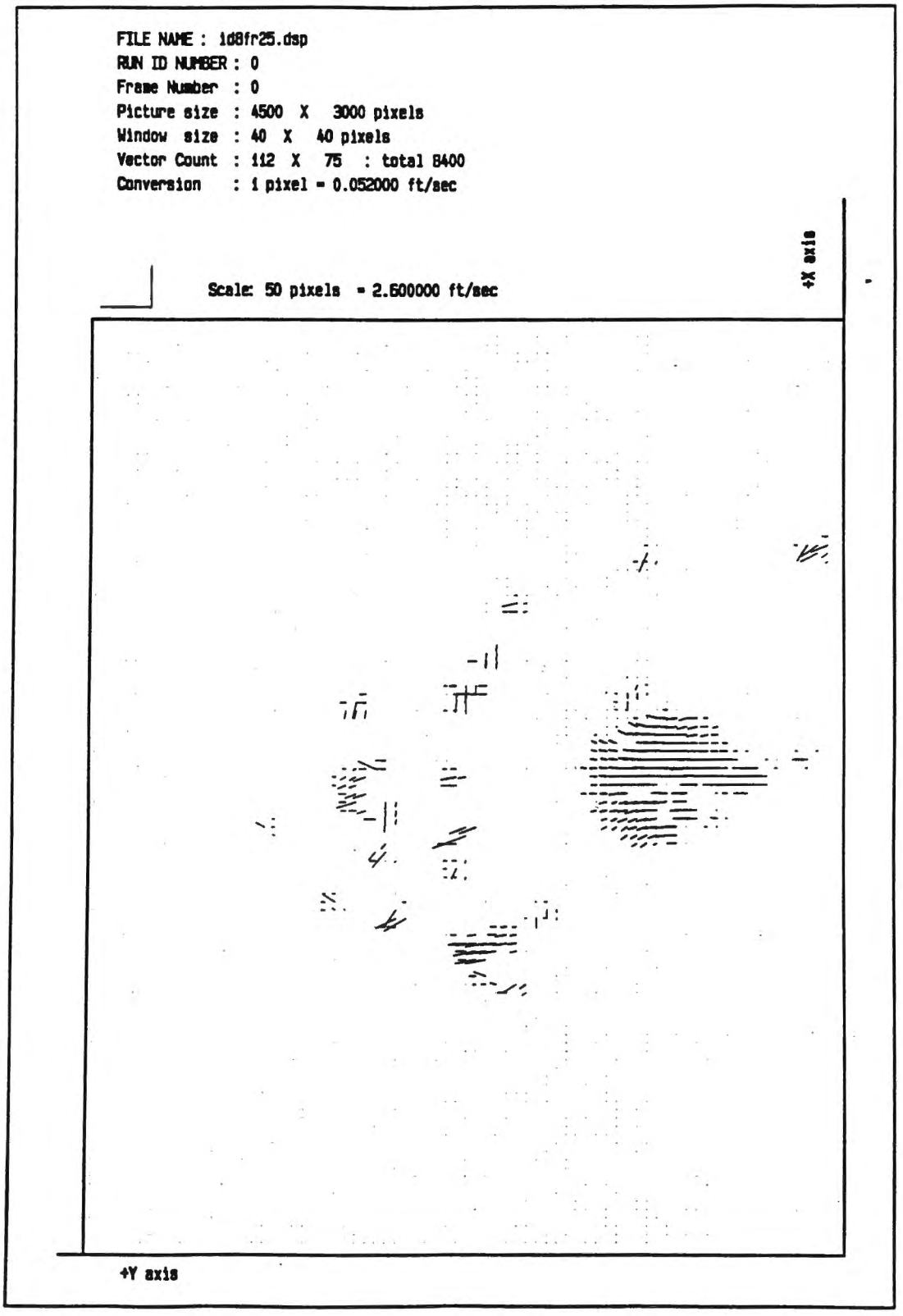

Figure 17. Startup Jet Flow, 2 kHz Laser Pulse (Reduced Data)

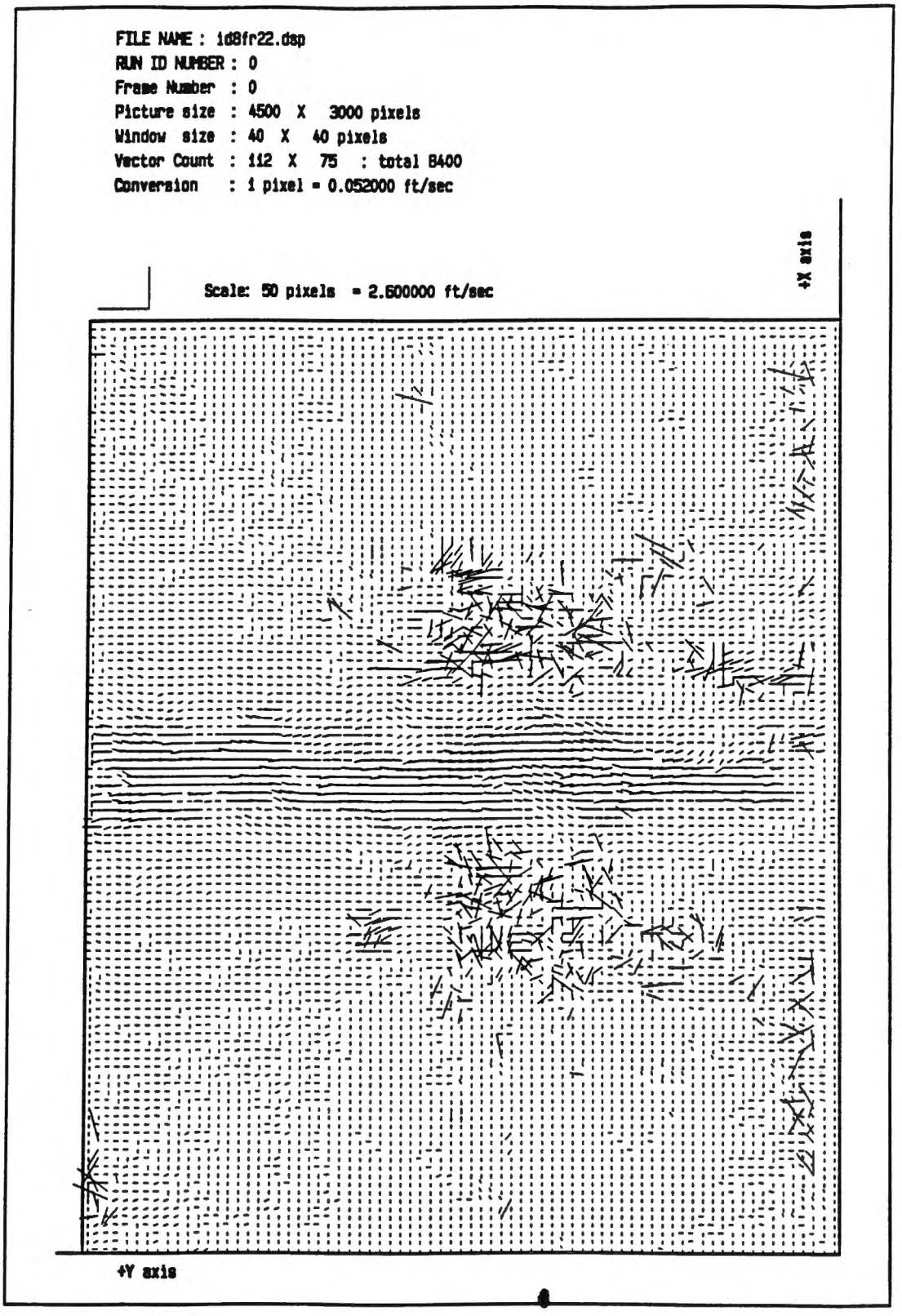

Figure 18. Steady State Jet Flow, 2 kHz Laser Pulse (Raw Data)

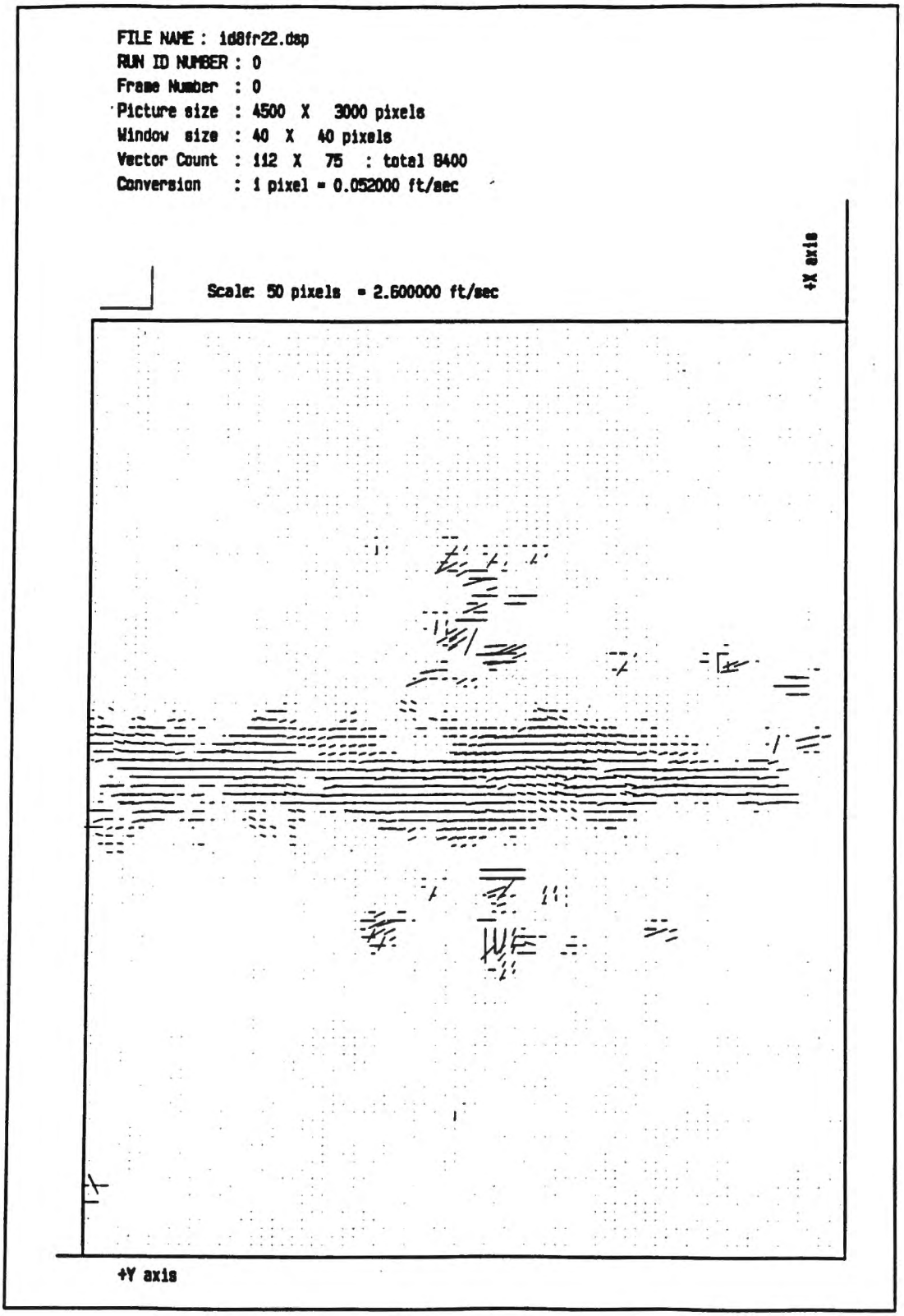

Figure 19. Steady State Jet Flow, 2 kHz Laser Pulse (Reduced Data)

All of the data show good performance at the high velocity regions of the flow. Since the testing was with low speed flow, some of the lower magnitude vectors fall below the minimum measurable velocity vector limit and are not displayed. The velocity limit is an artifact of the size of the particle image on the film and is discussed in the next section.

ACCURACY

The accuracy of the PIV system is dependent on a number of variables. The optical magnification and distortion can be measured by photographing a grid pattern in the test chamber. A good measurement of magnification is required to get an accurate vector displacement value. Accurate tracking of the flow can be best achieved by a neutrally buoyant 30 to 40 μ diameter particle. The low limit of measured velocity is the minimum possible measured displacement. Since the particles must be separated by at least one pixel, the minimum possible measured displacement is equal to the particle image diameter plus one. The vector window is sized by the maximum velocity, laser pulse frequency, and required accuracy.

Example:

Real Image Area: 10.8 cm x 7.2 cm = 4500 x 3000 pixels
Window Size: 0.12 cm x 0.12 cm = 50 x 50 pixels
Particle diameter = 3 pixels
Pixel ratio = 0.0024 cm/pixel
Number of laser pulses per frame = 2.

Maximum Flow Velocity: 10 m/sec = 1000 cm/sec
Set particle displacement for the maximum flow velocity to 40 pixels to fit within the 50 pixel wide window.
Measurement accuracy = 1 / particle displacement = 1 / 40 = 2.5% of span accuracy
For this case the measurement accuracy would be ± 12.5 cm/sec.

Laser Pulse Rate = flow rate / (pixel ratio x particle displacement)
= (1000 cm/sec / (0.0024 cm/pixel x 40 pixels) = 10,416 Hz.

Increasing the number of laser pulses per frame would require larger windows or higher laser pulse rates.

Minimum measurable velocity = (particle diameter + 1) x maximum flow rate / particle displacement = (3 +1) x 1000 cm/sec / 40 = 100 cm /sec.

This example show values for high vector density with reasonable accuracy. Larger windows may be required to get higher accuracy. Depending on the particle density in the water, larger windows may also be required to get 8 particle pair

images in each window. The scanning of the film is considered to be highly accurate. The computation of the vector by the FFT method can vary in accuracy depending on the computer system used.

CONCLUSIONS

Particle image velocimetry can provide the required global velocity field flow measurement tool required. Particle seeding requirements must be adjusted to each test. Laser pulse rates depend on the highest flow speed and required vector density. The lowest measurable displacement is also dependent on the laser pulse rate. Directional ambiguity of the vector is a trade-off with the vector density. Most of the foreseen work requires a high vector density.

RECOMMENDATIONS AND FUTURE WORK

While the current PIV system works, improvements can be made. The auto-loading system could be improved by cutting the film into slides and using an automatic slide loader on the scanner. The overall speed of the system could also be increased by reducing the transfer time to the Cray or doing the computations locally. A dedicated array processor on the PC would increased the efficiency of the system. Humphreys (reference 8) has shown that higher particle density could be achieved by using the computed average displacement vector for each window to find every particle pair displacement vector in the window. This technique could be added to the PIV software.

REFERENCES

1. J. S. Katz and T. T. Huang, "Quantitative Visualization of External and Internal Flows by Implementing the Particle Displacement Velocity Method," Experimental and Numerical Flow Visualization, FED-vol 128, ASME 1991.

2. R. J. Adrian, "Multi-Point Optical Measurements of Simultaneous Vectors in Unsteady Flow: A Review," Int. J. Heat Fluid Flow, 7,127 (1986).

3. "Kodak Black and White Films Data, Darkroom Handbook," Eastman Kodak Company, Rochester, NY, 14650.

4. "Kodak T-Max Professional Films," Kodak Publication No. F-32, Eastman Kodak Company, Rochester, NY, 14650.

5. R. D. Keane and R. J. Adrian, "Optimization of Particle Image Velocimeters with Multiple Pulse Systems," Proceedings of the 5th International Symposium on Applications of Laser Techniques to Fluid Mechanics, Lisbon, Portugal, paper 12.4, 1990.

6. R. J. Adrian, "Image Shifting Technique to Resolve Directional Ambiguity in Double-Pulsed Velocimetry," Applied Optics, vol. 25, no. 21, 1 November 1986.

7. C. C. Landreth and R. J. Adrian, "Electrooptical Image Shifting for Particle Image Velocimetry," Applied Optics, vol. 27, no. 20, 15 October 1988.

8. W. M. Humphreys, "A Histogram-Based Technique for Rapid Vector Extraction from PIV Photographs," Fourth International Conference on Laser Anemometry - Advances and Applications, Cleveland, OH, 1991.

Appendix A

LASER PULSE CONTROL CODE: LASER_CON.C

```
/* Laser Control Program for ESC-2000 on the */
/* Copper Vapor Laser                        */
/*        Kenneth LaPointe , NUWC, Code 8322        */
/*                last Rev Date : 4/23/92           */
/* Compile with cc laser_con.c -lmr -lgp -lm -o laser_con*/
        /* Camera sync pulse comes is TTL pluse and comes from
        the 35 mm camera. The sync pulse is connected to the "C"
        inputs of clocks 3,4 and 5. The output of 5 is connected into
        the "C" input of clock 6.
                Clock 3 outputs the laser start gate.
                Clock 4 outputs the laser stop gate.
                Clock 5 outputs a high level trigger to clock 6.
                Clock 6 outputs the laser pulse signals to the Laser
        */


#include </usr/include/mr.h>
#include <stdio.h>
#include <math.h>
#include <libmp.h>

long gls[SIZEOFGCA];

int i,j,k,l;
int clock_path[10];
int clk_da[10];
int nclocks;
int sixmhz = 11;                /* six Mhz rising edge                          */
int highlevelgate = 4;
int lowlevelgate = 5;
int risingedgegate = 6 ;/* Rising Edge Gate  , C input of this clock      */
int fallingedgegate = 7;
int startlow = 0;
int starthigh = 1;
int clockmode = 12;
int pulsemode = 1;
int countmode = 2;

float delay;
float laser_freq;
float number_pulses;
double high_volt = 3.50;
double low_volt = 2.0;
int msec_pause;
int laser_half_count;
char return_key[10];
static char *tname[] = {
                " ",
                "Laser Stop Pulse",
                "Laser Pulse Window",
                "Laser Shutter Opening",
                "Laser Start Gate",
                "Camera Shutter Opening",
                "Camera Ready Trigger",
```

```
                " "
                };
static char graph[] = {"trig.graph"};

double start_gate_width = 1000;
double stop_gate_width = 1000;
double start_gate_delay,stop_gate_delay;
double pulse_window_width, pulse_window_delay;
double camera_trigger_width = 1400;
double wret,dret;
int camera_ready_pretrig;
int lshut_time_to_open = 10000;
int high_pulse = 1;
int retrig_rising_edge = 3;

main()
{
system(" clear");
camera_ready_pretrig = 10100;
laser_freq = 2000.;

printf(" input a laser frequency : ");
scanf("%f",&laser_freq);


while (laser_freq >= 250 && laser_freq <= 10000)
{

mrclosall();
get_clocks_ready();

printf(" time for the laser shutter to open %d \n ",lshut_time_to_open);



        /* ALL TIMES ARE IN MICRO SECONDS */
        start_gate_delay = camera_ready_pretrig - lshut_time_to_open;
        if(start_gate_delay <= 0)
                {
                printf(" Start Gate Delay is negative !! \n");
                printf(" Increase camera pretrig level \n");
                exit(1);
                }

number_pulses = 3;


printf(" Delay time between Camera ready pulse and Start Pulse  : %f microsec
\n",start_gate_delay);
printf(" Laser External Driver Frequency (square wave)                  : %f Hz \n",laser_freq);
printf(" Number of visible pulses requested                             : %f \n",number_pulses);

printf(" Time it takes for the shutter to open %d \n ", lshut_time_to_open);
```

```
        /* Make sure to get the requred number of pulses */
        /* number of requested pulses plus 1 pre-ionization pulse and */
        /* room for all pulses to be completed */
        number_pulses = number_pulses + 1.05;


        set_clocks();
        show_data();


printf(" CLOCKS ARE RUNNING \n");


printf(" input a laser frequency : ");
scanf("%f",&laser_freq);

/*
printf(" Input Camera pretrig : ");
scanf("%d",&camera_ready_pretrig);
*/

system(" clear");
}

/*
printf(" Hit Return Key to Quit : ");
        scanf("%c",return_key);
*/
        mrclosall();

printf(" All Devices have Been Closed \n");

}


get_clocks_ready()
{
printf(" Get the clocks assigned and ready \n");
        nclocks = 4;
        for ( i = 0 ; i < nclocks ; ++i)
                {
                clock_path[i] = -1;
                clk_da[i] = -1;
                }

                mropen(&clock_path[0],"/dev/dacp0/clk3",0);
                mropen(&clock_path[1],"/dev/dacp0/clk4",0);
                mropen(&clock_path[2],"/dev/dacp0/clk5",0);
                mropen(&clock_path[3],"/dev/dacp0/clk6",0);

                mropen(&clk_da[0], "/dev/dacp0/clkda0", 0);
                mropen(&clk_da[1], "/dev/dacp0/clkda1", 0);
                mropen(&clk_da[2], "/dev/dacp0/clkda2", 0);
                mropen(&clk_da[3], "/dev/dacp0/clkda3", 0);
```

```
                mrclkdis(nclocks,clock_path);


printf(" set the trigger level to %f and %f volts \n",
                        high_volt, low_volt);
                mrclkdaset(clk_da[0],high_volt);
                mrclkdaset(clk_da[1],high_volt);
                mrclkdaset(clk_da[2],low_volt);
                mrclkdaset(clk_da[3],high_volt);

}
set_clocks()
{
printf(" set up the clocks with the requested values \n");


        /* Start Pulse output */
        mroneshot(clock_path[0],0,start_gate_delay,&dret,0,start_gate_width,
                &wret,high_pulse,retrig_rising_edge);

        /* Laser Trigger Gate */
        pulse_window_delay = start_gate_delay + lshut_time_to_open;
        pulse_window_width = number_pulses / laser_freq * 1000000;
        mroneshot(clock_path[2],0,pulse_window_delay,&dret,0,
                pulse_window_width,&wret,high_pulse,retrig_rising_edge);


        /* Put out the laser drive frequency , square wave */
        laser_half_count = 3000000.0 / laser_freq ;.
        mrclksetter(clock_path[3],sixmhz,laser_half_count,laser_half_count,
                highlevelgate,startlow,countmode,20);


        /* Stop Pulse Output */
stop_gate_delay = start_gate_delay + lshut_time_to_open + pulse_window_width  ;
        mroneshot(clock_path[1],0,stop_gate_delay,&dret,0,stop_gate_width,
                &wret,high_pulse,retrig_rising_edge);

printf(" Start gate delay (usec) : %f \n",start_gate_delay);
printf(" Start gate width (usec) : %f \n",start_gate_width);
printf(" Pulse Window delay (usec) : %f \n",pulse_window_delay);
printf(" Pulse Window width (usec) : %f \n",pulse_window_width);
printf(" Laser half count          : %d \n",laser_half_count);
printf(" Stop gate delay  (usec) : %f \n",stop_gate_delay);
printf(" Stop gate width  (usec) : %f \n",stop_gate_width);


        mrclkarm(nclocks,clock_path);

}

show_data()
{
```

```
float t[100];
float v[100];
float start = 0.0;
float offset ;
float cam_shut_open_time       ;
float cam_shut_rise_time;
float restart;

offset = 100;

restart = 1000000.0/ 25.0;
cam_shut_rise_time = 7000;
cam_shut_open_time = 1000;


        mpinit(gls);
        mpaxtype(gls,7,21);

        t[0] = -5000;
        /* Camera Trigger */
        v[0] = offset;
        t[1] = start;
        v[1] = offset;
        t[2] = t[1];
        v[2] = offset + 5;
        t[3] = start + camera_trigger_width;
        v[3] = offset + 5;
        t[4] = t[3];
        v[4] = offset;
        t[5] = start + restart;
        v[5] = offset;
        mplotsrcx( gls,1,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,1,6,0,v,"F",1,1,NULL,NULL);


offset = offset -10;
        /* Camera Shutter */
        v[0] = offset;
        t[1] = start + camera_ready_pretrig - cam_shut_rise_time;
        v[1] = offset;
        t[2] = t[1] + cam_shut_rise_time;
        v[2] = offset + 5;
        t[3] = t[2] + cam_shut_open_time;
        v[3] = offset + 5;
        t[4] = t[3] + cam_shut_rise_time;
        v[4] = offset;
        t[5] = start + restart ;
        v[5] = offset;
        mplotsrcx( gls,2,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,2,6,0,v,"F",1,1,NULL,NULL);

offset = offset -10;
        /* start Gate */
```

```
        v[0] = offset;
        t[1] = start + start_gate_delay;
        v[1] = offset;
        t[2] = t[1];
        v[2] = offset + 5;
        t[3] = t[2] + start_gate_width;
        v[3] = offset + 5;
        t[4] = t[3];
        v[4] = offset;
        t[5] = start + restart ;
        v[5] = offset;
        mplotsrcx( gls,3,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,3,6,0,v,"F",1,1,NULL,NULL);

offset = offset -10;
        /* laser Shutter */
        v[0] = offset;
        t[1] = start + start_gate_delay;
        v[1] = offset;
        t[2] = t[1] + lshut_time_to_open;
        v[2] = offset + 5;
        t[3] = t[2] + pulse_window_width;
        v[3] = offset + 5;
        t[4] = t[3] + lshut_time_to_open;
        v[4] = offset;
        t[5] = start + restart ;
        v[5] = offset;
        mplotsrcx( gls,4,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,4,6,0,v,"F",1,1,NULL,NULL);

offset = offset -10;
        /* laser Gate */
        v[0] = offset;
        t[1] = start + pulse_window_delay;
        v[1] = offset;
        t[2] = t[1];
        v[2] = offset + 5;
        t[3] = t[2] + pulse_window_width;
        v[3] = offset + 5;
        t[4] = t[3];
        v[4] = offset;
        t[5] = start + restart ;
        v[5] = offset;
        mplotsrcx( gls,5,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,5,6,0,v,"F",1,1,NULL,NULL);


offset = offset -10;
        /* stop  Gate */
        v[0] = offset;
        t[1] = start + stop_gate_delay;
        v[1] = offset;
        t[2] = t[1];
        v[2] = offset + 5;
```

```
        t[3] = t[2] + stop_gate_width;
        v[3] = offset + 5;
        t[4] = t[3];
        v[4] = offset;
        t[5] = start + restart ;
        v[5] = offset;

        mplotsrcx( gls,6,6,0,t,"F",1,1,NULL,NULL);
        mplotsrcy( gls,6,6,0,v,"F",1,1,NULL,NULL);

        mpaxvals(gls,1,-5000.0,UNDEF,5000.,-1);
        mpaxvals(gls,2,40.0,110.0,10.0,-1);

        mptitle(gls,1,-1,-1," 5 milliseconds per division");
        mplabel(gls,2,7,1,0.0,-1,-1,tname);


        mpfile(gls,graph,1,2);

        mplot(gls,0,1,0);
        mpend(gls);

}

transient_clock()
{
/* This part is for a varing laser frequency */

printf(" HIT THE RETURN KEY TO START RUN :: ");
        scanf("%c",return_key);
printf(" countdown starting \n");
                printf(" ... 5 ... \n");
                astpause(0,1000);
                printf(" ... 4 ... \n");
                astpause(0,1000);
                printf(" ... 3 ... \n");
                astpause(0,1000);
                printf(" ... 2 ... \n");
                astpause(0,1000);
                printf(" ... 1 ... \n");
                astpause(0,1000);

printf(" ... going with first clock... \n" );

        msec_pause = 500;
        astpause(0,msec_pause);


        laser_freq = laser_freq *2 ;
        set_clocks();
        astpause(0,msec_pause);
```

Appendix B

FILM PROCESSING CONTROL CODE: RUNPIV.C

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <math.h>

int i,j,k,l,m,n;
int start_frame, number_frames;
int tid_number;

char command_string[40];
FILE *fp,*fopen();

main()
{

system("cls");

printf(" Welcome to the Particle Image Velocimetry Computing System \n");
printf(" The system will compute the displacments of the particles \n\n");

printf(" Make sure the film is aligned ! \n\n");

printf(" Enter the Test ID number : ");
scanf("%d",&tid_number);

printf(" Enter the First frame number : ");
scanf("%d",&start_frame);

printf(" Enter the number of frames   : ");
scanf("%d",&number_frames);


for(i = start_frame; i < start_frame + number_frames; i++)
      {
      system("cls");

      printf(" First Digitize the picture \n");
         fp=fopen("command.bat","w");
         fprintf(fp," scanpic ID%dFr%d.img \n",tid_number,i);
         fclose(fp);
         system("command.bat");

      printf(" Send the picture to cray and submit the job \n");
         fp=fopen("command.bat","w");
         fprintf(fp," submit ID%dFr%d \n",tid_number,i);
         fclose(fp);
         system("command.bat");

     printf(" Advance the picture for the next frame \n");
         system("advance");

      }

}
```

Appendix C

SCANNER CONTROL CODE: SCANPIC.C

```
/*  Image Scanning program by Kenneth LaPointe Last REV 2/21/92 */
/* Reads in a monochrome image from Nikon Film Scanner        */
/* compile using Microsoft C, "cl scanpic.c mcib.obj"         */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "decl.h"

char   buffer[4600];  /* buffer array to transfer data using GPIB */
short  color[3200];   /* storage array for pixel grey scale data  */
long   buffer_length; /* total size of buffer to be transfered in */
long   buffer_out;    /* size of buffer (bytes) to be stored in file */
int    header = 4;    /* size of header in GPIB transfer (words ) */
int    etb = 1;       /* end of buffer word */
int    data_length;   /* length of total number of pixels in bytes */
int    pitch;         /* scanning skip ration                      */
int    width;         /* number of pixels in the x direction to scan */
int    height;        /* number of pixels in the y direction to scan */
int    status;        /* status of scanner , busy/ not busy     */
int    i,j,k,l,m,n,o; /* dummy variables */
char   file_name[30]; /* array to hold file name */

/* FILE VARIABLES */
int    datafile,n_write,n_read;

main(argc,argv)
int argc;
char *argv[];
{
/* check to see that the command line is proper */
     if( argc == 1 || argc >= 3){
          printf(" usage : scanpic output_filename.img  \n");
          exit(1); }

system("cls");
     datafile = open(argv[1],O_CREAT|O_TRUNC|O_BINARY|O_RDWR,0600);
     printf(" Data file : %s : has been opened \n",argv[1]);

printf(" Make the GPIB board number 0 the controller in charge \n");
     SendIFC(0);

printf(" Clear the Scanner , Device address is 1 \n");
     DevClear(0,1);

printf(" Set the scanner for monocrome film  reading a negative \n");
     Send(0,1,"MN\r",3L,NLend);

printf(" Auto Focus The Film .....\n");
     Send(0,1,"AF\r",3L,NLend);
     WaitSRQ(0, &status);

printf(" Set the scanner for pixel discharge \n");
```

```
    Send(0,1,"SPOF\r",5L,NLend);

printf(" Do a prescan ..... \n");
    Send(0,1,"PR\r",3L,NLend);
    WaitSRQ(0,&status);

printf(" Set the Exposure time Normal = 40  : \n");
    Send(0,1,"AP40\r",5L,NLend);

    printf(" Set the green filter for black and white \n");
    Send(0,1,"FG\r",3L,NLend);

    printf(" Requested a High Res Scan \n");
    high_res();


    printf(" Return scanner to zero position \n");
    Send(0,1,"MV0\r",4L,NLend);

    printf(" Close the Data file \n");
    close(datafile);

printf(" ..... Finished Scanning Data .....\n");
}


high_res()
{
printf(" READ HIGH RES 4500 x 3000 \n \n");
printf(" file size : 27 MB \n");
    /* Start at 250,1000 , End at  4749,3999 */
    /* area of box size 4500 x 3000 */
    /* pitch is 1 or every fifth point */
    /* number of data points 900 x 600 */
    width = 4500;
    height = 3000;
    buffer_out = height * 2;

    Send(0,1,"BX 250,1000,1,1,4500,3000,V\r",28L,NLend);
/*                              |           */
/*          12345678901234567890123456789012345678901234567890  */
    /* buffer length is size of header plus data + etb */
    /* need to do a read for each line (pixel array ) of data */
    /* plus one last blank column */
    buffer_length = header + height + etb;

printf(" READING DATA NOW ..... \n");
    for(j = 0 ; j < width;j++)
        {
        /* get a buffer full of data */
        Receive(0,1,buffer,buffer_length,STOPend);
        /* Convert the character data to integer data */
        i = 0;
        for( k = header; k < height + header; k++)
```

```
                {
                color[i] = buffer[k];
                if( color[i] < 0) color[i] =  color[i] + 256;
                i++;
                }
            n_write = write(datafile,color,buffer_out);
            }
printf(" Finished with data \n");
      /* empty the last blank buffer */
      Receive(0,1,buffer,5L,STOPend);

}

/* ========================================================== */

low_res()
{
printf(" READ LOW RESOLUTION 900 x 600 \n \n");
printf(" file size : 10 MB \n");
      /* Start at 250,1000 , End at  4749,3999 */
      /* area of box size 4500 x 3000 */
      /* pitch is 5 or every fifth point */
      /* number of data points 900 x 600 */
      width = 900;
      height = 600;
      buffer_out = height * 2;

      Send(0,1,"BX 250,1000,5,5,900,600,V\r",26L,NLend);
/*                              |             */
/*          12345678901234567890123456789012345678901234567890  */
      /* buffer length is size of header plus data + etb */
      /* need to do a read for each line (pixel array ) of data */
      /* plus one last blank column */
      buffer_length = header + height + etb;

printf(" READING DATA NOW ..... \n");
      for(j = 0 ; j < width;j++)
            {
            /* get a buffer full of data */
            Receive(0,1,buffer,buffer_length,STOPend);
            /* Convert the character data to integer data */
            i = 0;
            for( k = header; k < height + header; k++)
                  {
                  color[i] = buffer[k];
                  if( color[i] < 0) color[i] =  color[i] + 256;
                  i++;
                  }
            n_write = write(datafile,color,buffer_out);
            }
printf(" Finished with data \n");
      /* empty the last blank buffer */
      Receive(0,1,buffer,5L,STOPend);
}
```

```
/* ========================================================== */
```

Appendix D

FILM TRANSPORT CONTROL CODE: ADVANCE.C

```
/* Program : advance.c                    */
/* Last Rev : 8/26/92                     */
/* Written by Ken LaPointe Code 8322, NUWC     */
/*                                        */
/* This code runs the film movement stages. It  */
/* will move the film one frame at a time      */
/*                                        */
/* Compile the code by using the line          */
/*    cl advance.c pmc300.obj             */
/* Make sure that pmc300.obj and pmc300.h are   */
/* in the current directory, the pmc files come */
/* from the Newport software                   */

#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <math.h>

#include "PMC300.h"

#define x_axis 1
#define y_axis 2
#define r_axis 3
#define conv_mm 20000
#define conv_deg 1000

float frame_length = 38.5;

double samfreq= 488.25;
double gain_x = 50.0;
double pole_x = -0.1;
double zero_x = 0.8;
double maxvel_x = 0.40;
double accel_x = 10.0;
double gain_y = 50.0;
double pole_y = -0.1;
double zero_y = 0.8;
double maxvel_y = 0.40;
double accel_y = 10.0;
double gain_r = 10.0;
double pole_r = -0.1;
double zero_r = 0.7;
double maxvel_r = 8.00;
double accel_r = 10.00;

long x_act,y_act,r_act;

int i,j,k,l,m,n,o,p;
int frame_number;
int motion,stop;

FILE *fp,*fopen();

main()
```

```
{

/*** starting check ********/
initialize_hardware();

/******* move film ******/
move_film();

}

/***************** subroutines ***************/
/*********************************************/

move_film()
{
/* The starting position is close to the NIKON and */
/* away from the film                              */

/* Now Push the pins into the film */
                x_act = 5.0 * conv_mm;
                set_com_pos(x_axis,x_act);
                check_motion();

/* Move the Pins to Advance the film one frame */
                y_act = - frame_length * conv_mm;
                set_com_pos(y_axis,y_act);
                check_motion();

/* Pull the pins away from the film , the zero position */
                x_act = 0;
                set_com_pos(x_axis,x_act);
                check_motion();

/* Wind the Film a bit  in  degrees */
                r_act = -40 * conv_deg;
                set_com_pos(r_axis,r_act);
                check_motion();

/* Return the pins to the zero position for the next advance */
                y_act = 0;
                set_com_pos(y_axis,y_act);
}

check_motion()
{
        motion = 1;
        stop = 0;
        while(motion != stop)
                {
                moving();
                }
}
```

```
moving()
{
long counter1_old,counter2_old,counter3_old;
long counter1_new,counter2_new,counter3_new;
long counter;

int x_motion,y_motion,r_motion;


counter1_old = get_act_pos(x_axis);
counter2_old = get_act_pos(y_axis);
counter3_old = get_act_pos(r_axis);

for(counter = 0; counter < 100000;counter++)
        counter = counter;

counter1_new = get_act_pos(x_axis);
counter2_new = get_act_pos(y_axis);
counter3_new = get_act_pos(r_axis);

x_motion = 100;
y_motion = 10;
r_motion = 1;

if(counter1_old == counter1_new)
x_motion = 0;

if(counter2_old == counter2_new)
y_motion = 0;

if(counter3_old == counter3_new)
r_motion = 0;

motion = x_motion + y_motion + r_motion;

}

initialize_hardware()
{
        mc_init();
        config_ports(1,1,1,1);

/*** set up x axis *****/
        reset(x_axis);
        set_sample_freq(x_axis,samfreq,0);
        set_zero(x_axis,zero_x);
        set_pole(x_axis,pole_x);
        set_gain(x_axis,gain_x);
        clr_act_pos(x_axis);
        enter_ctl_mode(x_axis);
        set_prop_vel(x_axis,maxvel_x);

/*** set up y axis *****/
        reset(y_axis);
```

```
        set_sample_freq(y_axis,samfreq,0);
        set_zero(y_axis,zero_y);
        set_pole(y_axis,pole_y);
        set_gain(y_axis,gain_y);
        clr_act_pos(y_axis);
        enter_ctl_mode(y_axis);
        set_prop_vel(y_axis,maxvel_y);

/*** set up rotary axis *****/
        reset(r_axis);
        set_sample_freq(r_axis,samfreq,0);
        set_zero(r_axis,zero_r);
        set_pole(r_axis,pole_r);
        set_gain(r_axis,gain_r);
        clr_act_pos(r_axis);
        enter_ctl_mode(r_axis);
        set_prop_vel(r_axis,maxvel_r);
}
```

Appendix E

CRAY JOB CONTROL CODE: SUBMIT.BAT

```
@echo off
echo Script to send the Image file to the CRAY
echo and compute the displacement vectors
echo the data will be sent to the PI83a computer

echo Make the Job script
     echo rm %1.e* %1.o*                > %1.job
     echo cd /usr/tmp/PIV               >> %1.job
     echo rm %1.act                     >> %1.job
     echo rm %1.dsp                     >> %1.job
     echo ja                            >> %1.job
     echo fftpiv %1.img %1.dsp          >> %1.job
     echo ja -st                        >> %1.job
     echo %1.snd                        >> %1.job
     echo rm %1.img                     >> %1.job
     echo rm %1.job                     >> %1.job
     echo rm %1.snd                     >> %1.job

echo Make The Cray Send the info back to the SGI
     copy send.h %1.snd
     echo put %1.dsp       >> %1.snd
     echo put %1.act       >> %1.snd
     echo cd               >> %1.snd
     echo mput %1.*        >> %1.snd
     echo EOF              >> %1.snd

echo SET UP THE PC TO SEND THE INFO TO THE CRAY
     copy C:\ENET\cray.log %1.put
     echo prompt                        >> %1.put
     echo verbose                       >> %1.put
     echo ascii                         >> %1.put
     echo cd /usr/tmp/PIV               >> %1.put
     echo put %1.job                    >> %1.put
     echo put %1.snd                    >> %1.put
     echo binary                        >> %1.put
     echo put %1.img                    >> %1.put
     echo bye                           >> %1.put
     C:\PATHWAY\ftp < %1.put

echo Make the %1.snd file exacutable
     rsh cray chmod 777 /usr/tmp/PIV/%1.snd

echo .
echo .

echo ******** check on files sent *******
     C:\PATHWAY\rsh cray ls -al /usr/tmp/PIV

echo Submit the batch Job to the Cray
     rsh cray qsub -lT 00:08:00 -lm 2.5MW -r %1 /usr/tmp/PIV/%1.job
     rsh cray qstat -ba

echo remove the files on the PC
     del %1.img
```

```
del %1.snd
del %1.put
del %1.job
```

Appendix F

AUTOCORRELATION PIV CODE: AUTOCORE.C

```
/**    PARTICLE IMAGE VELOCEMETRY PROGRAM      **/
/**       using the autorcorrelation method      **/
/**    By :   Kenneth LaPointe              **/
/**          NUSC, Code 8322              **/
/**    Last rev as of  AUG 17 1992          **/
/** This program reads an image                              **/
/** files to use an auto correlation program    **/
/** to compute the displacemnet between        **/
/** particles                          **/
/** compile on sgi  as : cc -O3 autocore.c -Dsgi  -lm -o autocore */
/** compile on cray as : cc -O3 autocore.c -Dcray -lm -o autocore        */

/** Input for the program is '                                    */
/**       autocore filename.img filename.dsp filename.up                                  */
/** the .img file is the input file scanned by the NIKON scanner        */
/** the .dsp file is the output file computed by the program            */
/** the .up is the setup file for the code                              */

#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>

int true = 1;
int false = 0;
int i,j,k,l,m,n,o,p;
float a,b,c,d,e,f,g;

int junk;
int nbuff;
int buffer_bytes;
int buf_loop,x_offset;
int pic_width;
int win_rel;

short rgbvector[3];
long datastore;

int xw1,yw1;   /* lower left corner of 2 D array is xw1,yw1          */
int xw2,yw2;   /* upper right corner of 2 D array is xw2,yw2         */
int csx,csy;   /* copy shift value in x and y, is the amount the copy is moved */
int start_csx,start_csy; /* amount of initial movment of the copy      */
int end_csx,end_csy;    /* move the copy to this point              */
int hold_csy;  /* hold buffer varible for the csy value             */
int hold_csx;
int isx,isy;   /* 2D image array index start point in x and y        */
int iey,iex;   /* 2D iamge array index end point in x and y          */


int iix,iiy;   /* 2D image array index x,y                          */
int cix,ciy;   /* 2D copy array index x,y,                          */
int range_x,range_y;

int xw_width,yw_width;
```

```
/* for the correlatons, there are three positions,     */
int pz = 2;    /* number 2 is the zero position                */
int pn = 1;    /* number 1 is the minus position                */
int pp = 0;    /* number 0 is the positive position             */
int xp = 0;    /* and the x and y values                     */
int yp = 1;
int disp_peak[3][2];
int scan_area;
float corr_hist[2][200][200];
float corr_peak[3];

int n_windows,win;
int max_frames,run_number;
int x_win_step,y_win_step;
float total;

int max_x_disp,max_y_disp;

char file_name[20];          /* file name character storage */
char last_data_file[20];



int image_file;              /* file pointer for image file       */
int height,width;

int height;
char old_disp_name[50];
char new_disp_name[50];
int shade;
int n_wind_sec_x, n_wind_sec_y;
int frame_number;
int dia_part;
int xwin,ywin;
float skipx, skipy;

int range_x_guess, range_y_guess;

int cs_csx,ce_csx,cs_csy,ce_csy;
int draw_data;
int shift;


float x_sum,y_sum,x_weight,y_weight;
float x_sum_total,y_sum_total;

int shade_limit;

#ifdef cray
unsigned short image_data[512][3200];
long buffer[1024];
unsigned long s1,s2,s3,s4;
off_t seek_error,image_offset_bytes;
int max_array_width = 500;
```

```
int max_array_height = 3000;
int  peak_value[500][2];
float avg_value[500][2];
float zero_corr_ratio[500];
#endif

#ifdef sgi
unsigned short image_data[512][3200];
short buffer[3100];
off_t seek_error,image_offset_bytes;
int max_array_width = 500;
int max_array_height = 3000;
int  peak_value[500][2];
float avg_value[500][2];
float zero_corr_ratio[500];
#endif

int fd,n_read,n_write,bytes;          /* file descriptors */
FILE *file_dat,*look,*sfile,*new_file,*old_file,*fopen();

/* ************************************************************** */

main(argc,argv)
int argc;
char *argv[];
{

/* ====================== REQUIRED INPUTS ================== */
/* A 35 mm negitive  is 36mm x 26 mm , with the Nikon scanner at a pitch of 1 */
/* the negative is digitized to 125 pixels/mm or 4500 x 3000 pixels */
/* For use on the Cray the size of the buffer and picture  must be an integer */
/* multiple of 4 */

/** read in the set up file */
sfile = fopen(argv[3],"r");

fscanf(sfile,"%d",&height);
fscanf(sfile,"%d",&width);
fscanf(sfile,"%d",&xw_width);
fscanf(sfile,"%d",&yw_width);
fscanf(sfile,"%d",&shade_limit);
fscanf(sfile,"%d",&max_x_disp);
fscanf(sfile,"%d",&max_y_disp);
fscanf(sfile,"%d",&range_x_guess);
fscanf(sfile,"%d",&range_y_guess);
fscanf(sfile,"%d",&dia_part);

fclose(sfile);

/*
Standard settings are
height = 3000;
width = 4500;
xw_width = 80;
```

```
yw_width = 80;
shade_limit = 10;
max_x_disp = 40;
max_y_disp = 40;
range_x_guess = 30;
range_y_guess = 30;
dia_part = 12;
*/


printf(" height %d \n",height);
printf(" width %d \n",width);
printf(" xw width %d \n",xw_width);
printf(" yw width %d \n",yw_width);
printf(" shade limit %d \n",shade_limit);
printf(" max x disp %d \n",max_x_disp);
printf(" max y disp %d \n",max_y_disp);
printf(" range x %d \n",range_x_guess);
printf(" range y %d \n",range_y_guess);
printf(" dia part %d \n",dia_part);


if(height > max_array_height )
        {
        printf(" height too big \n");
        exit(1);
        }


#ifdef sgi
if(width > max_array_width )
        {
                pic_width = width;
                width = max_array_width - max_x_disp - range_x_guess;
        }
#endif

#ifdef cray
        if( width > max_array_width)
                {
                pic_width = width;
                width = xw_width;
                }

#endif




/* for the each input buffer */
n_wind_sec_y = height / yw_width;
n_wind_sec_x = width  / xw_width; /* number of windows in the x direction */
n_windows = n_wind_sec_x * n_wind_sec_y;/* total number of windows and vectors  */
```

```
nbuff = (pic_width / xw_width)/ n_wind_sec_x;

printf(" max disp        %d \n",max_x_disp);
printf(" range x guess  %d \n",range_x_guess);
printf(" dia part        %d \n",dia_part);
printf(" n sec y  %d \n",n_wind_sec_y);
printf(" n sec x  %d \n",n_wind_sec_x);
printf(" n windows      %d \n",n_windows);
printf(" number of buffers  %d \n",nbuff);


/*** open the Image data file ***/

image_file = open(argv[1],0);

/*** open the displacment data file **/
new_file = fopen(argv[2],"w");
fprintf(new_file,"%d \n",run_number);
fprintf(new_file,"%d \n",frame_number);
fprintf(new_file,"%d \n",width * nbuff);
fprintf(new_file,"%d \n",height);
fprintf(new_file,"%d \n",n_wind_sec_x * nbuff);
fprintf(new_file,"%d \n",n_wind_sec_y);


x_offset = 0;

#ifdef cray
/* compute the number of buffers */
buffer_bytes = height * 2;

for (buf_loop = 0; buf_loop < nbuff; buf_loop++)
        {

/******** for testing *******/

        image_offset_bytes = buf_loop * xw_width * n_wind_sec_x  * buffer_bytes;
        get_image_file();
        piv();
        dump_disp();
        }
#endif

#ifdef sgi
/* compute the number of buffers */
buffer_bytes = height * 2;

for (buf_loop = 0; buf_loop < nbuff; buf_loop++)
        {
        image_offset_bytes = buf_loop * xw_width * n_wind_sec_x  * buffer_bytes;
        get_image_file();
        piv();
        dump_disp();
        }
```

```
#endif

/* close the files */
fclose(new_file);
close(image_file);

}



#ifdef cray
get_image_file()
{
int end_loop;

/* Read in all of the data line by line                    */
/* the data is read into the array " buffer"                */
/* Since the data went from a 16 bit machine (PC) to a 64 bit machine (CRAY) */
/* four data points are in one CRAY word */

/*
printf(" buf loop %d  width %d   buf bytes %d image offset %ld  xoffset %d \n",
buf_loop,width,buffer_bytes,image_offset_bytes,x_offset);
*/


seek_error = lseek(image_file,image_offset_bytes,SEEK_SET);

end_loop = 2 * width;

if( buf_loop + 1 == nbuff)
                end_loop = xw_width * n_wind_sec_x;

for ( j = 0 ; j < end_loop ; ++j)
        {
        m = 0;
        n_read = read(image_file,buffer,height * 2);
        for( k = 0 ; k < height/4; k++)
                {
image_data[j][m] = image_data[j][m+1] = image_data[j][m+2] = image_data[j][m+3] = 0;
        s1 = buffer[k] ;
        image_data[j][m ] = s1 >> 56;
        if(image_data[j][m] < shade_limit | | image_data[j][m] > 255)    image_data[j][m] = 0;

        s2 = buffer[k] << 16;
        image_data[j][m+1] = s2 >> 56;
        if(image_data[j][m+1] < shade_limit | | image_data[j][m+1] > 255) image_data[j][m+1] = 0;

        s3 = buffer[k] << 32;
        image_data[j][m+2] = s3 >> 56;
        if(image_data[j][m+2] < shade_limit | | image_data[j][m+2] > 255) image_data[j][m+2] = 0;

        s4 = buffer[k] << 48;
        image_data[j][m+3] = s4 >> 56;
```

```
        if(image_data[j][m+3] < shade_limit || image_data[j][m+3] > 255) image_data[j][m+3] = 0;


                m +=4;
                }

        }

}
#endif



#ifdef sgi
get_image_file()
{
int end_loop;
printf(" reading data .\n");
/* Read in all of the data line by line                                    */
/* the data is read into the array " buffer"                               */
/* Since the data went from a 16 bit machine (PC) to a 32 bit machine (SGI) */
/* each element of the array is bit shifted by 8 bits                      */
/* The data on the PC is 1's complement, the data on the SGI is 2's        */
/* complement, so if the data is less than 0 add 256                       */


seek_error = lseek(image_file,image_offset_bytes,SEEK_SET);

if(seek_error == -1)
        exit(1);

end_loop = width;

if( buf_loop -1 == nbuff)
                end_loop = xw_width * n_wind_sec_x;
for ( j = 0 ; j < end_loop ; ++j)
        {
        n_read = read(image_file,buffer,height * 2);
        for( k = 0 ; k < height; ++k)
                {
                shade = buffer[k] >> 8;
                if( shade < 0   ) shade += 256;
                if( shade < shade_limit   ) shade = 0;
                image_data[j][k] = shade;
                }
        }

}

#endif



dump_disp()
```

```
{
win = 1;
for (x_win_step = 0 ; x_win_step < n_wind_sec_x; x_win_step++)
        {
        xw1 = x_win_step * xw_width + x_offset;
        xw2 = xw1 + xw_width;
        for (y_win_step = 0 ; y_win_step < n_wind_sec_y; y_win_step++)
                {
                yw1 = y_win_step * yw_width;
                yw2 = yw1 + yw_width;

fprintf(new_file,"%d %d %d %d %d %d %d %f %f %f\n",win_rel,xw1,yw1,xw2,yw2,
        peak_value[win][xp],peak_value[win][yp],avg_value[win][xp],
                avg_value[win][yp],
        zero_corr_ratio[win]);

printf("%d %d %d %d %d %d %d %f %f \n",win_rel,xw1,yw1,xw2,yw2,
        peak_value[win][xp],peak_value[win][yp],avg_value[win][xp],
        avg_value[win][yp]);

                win++;
                win_rel++;
                }
        }
x_offset = xw2;

}


piv()
{
xw1 = 0;      /* starting point of window in x direction (in pixels)      */
yw1 = 0;      /*                          y                               */
win = 1;      /* first window number                                      */

/* in This routine loop through each window, get the two corners of the window  */
/* use the maximumm estamated displacment to compute the auto-correlation       */

for (x_win_step = 0 ; x_win_step < n_wind_sec_x; x_win_step++)
        {
        xw1 = x_win_step * xw_width;
        xw2 = xw1 + xw_width;
        range_x = max_x_disp;
        range_y = max_y_disp;
        for (y_win_step = 0 ; y_win_step < n_wind_sec_y; y_win_step++)
                {
                yw1 = y_win_step * yw_width;
                yw2 = yw1 + yw_width;

printf(" ****** win %d xw1 %d xw2 %d   yw1 %d  yw2 %d *** \n",
                win,xw1,xw2,yw1,yw2);

                auto_corr();
```

```
printf("%d %d %d %d %d %d %d %f %f \n",win_rel,xw1,yw1,xw2,yw2,
        peak_value[win][xp],peak_value[win][yp],avg_value[win][xp],
        avg_value[win][yp]);

        range_x = range_x_guess;
        range_y = range_y_guess;
                win++;
                }
        }
}



auto_corr()
{
/* get the best possible auto_correlation value, the zero position */
perform_zero_corr();

/* Scan through the positive x and y dispacments */
perform_pos_corr();

/* Scan through the positive x and negative y dispacments */
perform_neg_corr();



printf(" POS DISPLACMENT DATA max value %.2f at : x %d  y %d  \n",
corr_peak[pp],disp_peak[pp][xp],disp_peak[pp][yp]);


printf(" NEG DISPLACMENT DATA max value %.2f at : x %d  y %d  \n",
corr_peak[pn],disp_peak[pn][xp],disp_peak[pn][yp]);


avg_value[win][xp] = 0;
avg_value[win][yp] = 0;
peak_value[win][xp] = 0;
peak_value[win][yp] = 0;


        if( corr_peak[pp] > corr_peak[pn] )
                {
                /* displacment is positive y*/
                scan_area = pp;
                zero_corr_ratio[win] = corr_peak[pp] /corr_peak[pz];
                scan_hist();
                }
        else
                {
                /* displacment is negative y*/
                scan_area = pn;
                zero_corr_ratio[win] = corr_peak[pn] /corr_peak[pz];
                scan_hist();
```

```
                avg_value[win][yp] = - avg_value[win][yp];
                }

}



perform_zero_corr()
{
/*  upper right quad */
corr_peak[pz] = 0.0001;
disp_peak[pz][xp] = 0;
disp_peak[pz][yp] = 0;

/** Copy is placed on top of the image , zero position**/
total = 0;
isy = yw1  ;
iey = yw2 ;
isx = xw1 ;
iex = xw2 ;

/** perform auto correleation for this copy placecent */
ciy = isy;
for( iiy = isy; iiy < iey; iiy++)
        {
        cix = isx;
        for( iix = isx; iix < iex; iix++)
                {

                total += image_data[iix][iiy] * image_data[cix][ciy];
                cix++;
                }
        ciy++;
        }
corr_peak[pz] = total;

}

perform_pos_corr()
{
/*  upper right quad +x , +y movment of the copy*/
corr_peak[pp] = 0;
disp_peak[pp][xp] = 0;
disp_peak[pp][yp] = 0;

printf(" Set up the positive corr parameters \n");

/* Set up the movment positions of the copy on the original          */
/* make sure the copy does not do the zero position and              */
/* does not go outside the bounds of the array memory locations      */
/* set the dispacment starting and stopping points based on a best   */
/* guess and some resonable range estimate */

start_csy = abs(peak_value[win -1][yp]) - range_y ;
```

```
if(start_csy < 0 ) start_csy = 0;

end_csy =  abs(peak_value[win -1][yp]) + range_y;
if(end_csy > height )end_csy = height;

start_csx = abs(peak_value[win -1][xp]) - range_x;
if(start_csx < 0 ) start_csx = 0;

end_csx =   abs(peak_value[win -1][xp]) + range_x;
if(end_csx > width) end_csx = width;

/* Remember not to do the zero position, The zero position is actually  */
/* the same size as a particle is in pixels                             */
/* If the copy is moved over the 0,0 position                           */
/* do the correlation in two steps                                      */
/* to the left and top of the partilcle and then all area above it      */

printf(" requested positive corr start csy %d end %d   start csx %d end %d \n",
        start_csy,end_csy,start_csx,end_csx);

if(start_csy <= dia_part && start_csx <= dia_part )
        {

        hold_csy = start_csy;
        start_csy = dia_part;
        hold_csx = end_csx;
        end_csx = dia_part;
        pos_copy_shift();

        start_csy = hold_csy;
        start_csx = dia_part;
        end_csx = hold_csx;
        pos_copy_shift();

        }
else
        {
        pos_copy_shift();
        }

}


pos_copy_shift()
{

printf(" actual positive corr start csy %d end %d   start csx %d end %d \n",
        start_csy,end_csy,start_csx,end_csx);

/** shift copy in x and y **/

/* Place the copy in the requested starting position          */

for(csy = start_csy; csy < end_csy; csy++)
```

```
        {
        isy = yw1 + csy ;
        iey = yw2 + csy ;
        for(csx = start_csx; csx < end_csx; csx++)
                {
                total = 0;
                isx = xw1 + csx ;
                iex = xw2 + csx ;
                ciy = yw1;
                for( iiy = isy; iiy < iey; iiy++)
                        {
                        cix = xw1;
                        for( iix = isx; iix < iex; iix++)
                                {


                total += image_data[iix][iiy] * image_data[cix][ciy];

                                cix++;
                                }
                        ciy++;
                }

/*
if(win_rel == 60 && win == 41)
printf(" %d %d total value is %f \n",csx,csy,total);
*/

                corr_hist[pp][csx][csy] = total;
                if(total > corr_peak[0])
                        {
                        corr_peak[pp] = total;
                        disp_peak[pp][xp] = csx;
                        disp_peak[pp][yp] = csy;
                        }
                } /* end loop csx */
        } /* end loop csy */

printf(" Finished pos auto corr \n");

}



perform_neg_corr()
{
/*  lower right quad */
corr_peak[pn] = 0;
disp_peak[pn][xp] = 0;
disp_peak[pn][yp] = 0;
printf(" Set up the neg corr parameters \n");

start_csy = -abs(peak_value[win -1][yp]) - range_y ;
```

```
if(start_csy < - height ) start_csy = - height;

end_csy =  -abs(peak_value[win -1][yp]) + range_y;
if(end_csy >= 0  )end_csy =  0;

start_csx = abs(peak_value[win -1][xp]) - range_x;
if(start_csx < 0 ) start_csx = 0;

end_csx =   abs(peak_value[win -1][xp]) + range_x;
if(end_csx > width) end_csx = width;

printf(" requested negative corr start csy %d end %d   start csx %d end %d \n",
        start_csy,end_csy,start_csx,end_csx);

if(end_csy  >=  -dia_part && start_csx <= dia_part )
        {
        hold_csy = end_csy;
        end_csy = - dia_part;
        hold_csx = end_csx;
        end_csx = dia_part;
        neg_copy_shift();

        end_csy = hold_csy;
        start_csx = dia_part;
        end_csx = hold_csx;

        neg_copy_shift();
        }
else
        {
        neg_copy_shift();
        }
}


neg_copy_shift()
{

printf(" actual negative corr start csy %d end %d   start csx %d end %d \n",
        start_csy,end_csy,start_csx,end_csx);

/** shift copy in x and y **/
/* rember to use the absolut values for the array index */
for(csy = start_csy; csy < end_csy; csy++)
        {
        isy = yw1 ;
        iey = yw2 - abs( csy );
        for(csx = start_csx; csx < end_csx; csx++)
                {
                total = 0;
                isx = xw1 + csx ;
                iex = xw2 + csx ;
                ciy = yw1 + abs(csy);
                for( iiy = isy; iiy < iey; iiy++)
```

```
                    {
                    cix = xw1;
                    for( iix = isx; iix < iex; iix++)
                            {

                total += image_data[iix][iiy] * image_data[cix][ciy];
                            cix++;
                            }
                    ciy++;
                    }
                corr_hist[pn][csx][abs(csy)] = total;
                if(total > corr_peak[pn])
                    {
                    corr_peak[pn] = total;
                    disp_peak[pn][xp] = csx;
                    disp_peak[pn][yp] = csy;
                    }
                }
        }

printf(" Finished neg auto corr \n");

}



scan_hist()
{
/* The histogram yields a peak which can be used to compute the displacment vector */
/* A better vector would be obtained by computing the average displacement around  */
/* the peak. Compute the average by getting the weighted value in each dimension */
/* Scan the correlation map and compute the average value around the peak */

x_sum = 0;
y_sum = 0;
x_sum_total = 1;
y_sum_total = 1;
x_weight = 0;
y_weight = 0;

        cs_csx = disp_peak[scan_area][xp] - 4;
if (cs_csx < 0) cs_csx = 0;
        ce_csx = disp_peak[scan_area][xp] + 4;

        cs_csy = disp_peak[scan_area][yp] - 4;
        ce_csy = disp_peak[scan_area][yp] + 4;

/* compress the x axis to one dimension             */
/* and weight the displacment by the vauel of the peak  */
/* compute the average displacment in the x direction   */

for(csx = cs_csx; csx <= ce_csx; csx++)
        {
        for(csy = cs_csy; csy <= ce_csy; csy++)
```

```
                {
                x_sum += corr_hist[scan_area][csx][abs(csy)];
                }
        x_weight += csx * x_sum;
        x_sum_total += x_sum;
        x_sum = 0;
        }
avg_value[win][xp] = x_weight/x_sum_total;


/* compress the y axis to one dimension                */
for(csy = cs_csy; csy <= ce_csy; csy++)
        {
        for(csx = cs_csx; csx <= ce_csx; csx++)
                {
                y_sum += corr_hist[scan_area][csx][abs(csy)];
                }
        y_weight += csy * y_sum;
        y_sum_total += y_sum;
        y_sum = 0;
        }

avg_value[win][yp] = y_weight/y_sum_total;

peak_value[win][xp] = disp_peak[scan_area][xp];
peak_value[win][yp] = disp_peak[scan_area][yp];
}
```

Appendix G

2D FFT PIV CODE FOR THE SGI: SGI_FFTPIV.C

```
/**    PARTICLE IMAGE VELOCEMETRY PROGRAM     **/
/**    By :   Kenneth LaPointe              **/
/**          NUSC, Code 8322              **/
/**    Last rev as of  May 5 1992           **/
/** This program reads in data files and image  **/
/** files to use an 2D FFT program    **/
/** to compute the displacemnet between        **/
/** particles                          **/
/** compile on sgi as : cc -O3 sgi_fftpiv.c -lm -o sgi_fftpiv          */
/* To see the IMAGE, and FFT's on the screen compile with */
/** compile on sgi as : cc -O3 sgi_fftpiv.c -Dshow -lgl -lm -o sgi_fftpiv    */

#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>
#ifdef show
#include <gl/gl.h>
#endif

int try;
int end_buffer;
int fft_loop ;
int fft_pass;
int true = 1;
int false = 0;
int i,j,k,l,m,n,o,p;
int aj1,aj2,aj3,ak1,ak2,ak3;
float a,b,c,d,e,f,g;
float cp;
double square, magnitude;

float fft_peak;
float fft_ratio;
float cen_ratio;
float dc_com;

float first_peak_value;
int first_peak_x;
int first_peak_y;

float second_peak_value;
int second_peak_x;
int second_peak_y;
int aindex;

int junk;
int nbuff;
int buffer_bytes;
int buf_loop,x_offset,start_buffer;
int pic_width;
int win_rel;

short rgbvector[3];
```

```
long datastore;

int xw1,yw1;   /* lower left corner of 2 D array is xw1,yw1        */
int xw2,yw2;   /* upper right corner of 2 D array is xw2,yw2       */

int xw_width,yw_width;

int n_windows,win;
int max_frames,run_number;
int x_win_step,y_win_step;
int total;

int xw1out,xw2out;


int max_x_disp,max_y_disp;

char file_name[20];          /* file name character storage */
char last_data_file[20];



int image_file;
int image_height,image_width;

char old_disp_name[50];
char new_disp_name[50];
int shade;
int n_wind_sec_x, n_wind_sec_y;
int frame_number;
int dia_part;
int xwin,ywin;
float skipx, skipy;

float cp;
int draw_data;
int shift;

int  peak_value_x;
int  peak_value_y;
float avg_value_x;
float avg_value_y;

float x_sum,y_sum,x_weight,y_weight;
float x_sum_total,y_sum_total;
float zero_limit;

int shade_low,shade_high;


short buffer[3100];
float data[200000];
int array_index = 256;
float fft_data[256][256];
```

```
off_t seek_error,image_offset_bytes;
unsigned short image_data[250][3200];
int max_array_width = 250;
int max_array_height = 3000;

int win_buf;

float hist[260],hist_scale,max_hist;
int  skip;
int buf_start;

float vscale;
int cx1,cx2;
int cy1,cy2;

int fd,n_read,n_write,bytes;          /* file descriptors */
FILE *sfile,*new_file,*old_file,*fp,*file_in,*fopen(),*fft_data_file;

/* ************************************************************** */
int isign;


int nn[4];
/* ************************************************************** */

/******** graphics window id's ***/
long image_win,fft1_win,fft2_win,ana_win;

main(argc,argv)
int argc;
char *argv[];
{

#ifdef show
foreground();
#endif
/* ====================== REQUIRED INPUTS ============================       */
/* A 35 mm negitive  is 36mm x 26 mm , with the Nikon scanner at a pitch of 1 */
/* the negative is digitized to 125 pixels/mm or 4500 x 3000 pixels          */
/* For use on the Cray the size of the buffer and picture  must be an integer        */
/* multiple of 4 */

image_height = 3000;
image_width = 4500;
shade_low = 10;
shade_high = 240;

#ifdef show
printf(" enter Low limit of the picture : ");
scanf("%d",&shade_low);

printf(" Enter the High limit of the picture : ");
```

```
scanf("%d",&shade_high);
#endif

/* The program must know the diameter of a particle so that the */
/* zero  position is skipped */
dia_part = 6;

/* ============================================================ */
/* compute the number of windows to break the image into. This will be the number */
/* of displacecent vectors per image ,  set the number so that there are        */
/* about 8 particle per window */

/* At this time the value must be either 16,32,64,128,or 256 */
xw_width = yw_width = 64;
nn[1] = 128;
nn[2] = 128;

if( nn[1] > array_index || nn[2] > array_index)
        {
        printf(" not enough memory in fft arrays \n");
        exit(1);
        }


if(image_height > max_array_height )
        {
        printf(" height too big \n");
        exit(1);
        }



if(xw_width > max_array_width )
        {
        printf(" xw_width too big \n");
        exit(1);
        }


/* for the each input buffer */
n_wind_sec_y = image_height / yw_width;
n_wind_sec_x = image_width  / xw_width; /* number of windows in the x direction */
n_windows = n_wind_sec_x * n_wind_sec_y;/* total number of windows and vectors  */

/*** open the Image data file ***/
image_file = open(argv[1],0);

/*** open the displacment data file **/
new_file = fopen(argv[2],"w");

fprintf(new_file,"%d \n",run_number);
fprintf(new_file,"%d \n",frame_number);
fprintf(new_file,"%d \n",image_width);
fprintf(new_file,"%d \n",image_height);
```

```
fprintf(new_file,"%d \n",xw_width);
fprintf(new_file,"%d \n",yw_width);
fprintf(new_file,"%d \n",n_wind_sec_x);
fprintf(new_file,"%d \n",n_wind_sec_y);
fprintf(new_file,"%d \n",nn[1]);
fprintf(new_file,"%d \n",nn[2]);

#ifdef show
vscale = 5;

cx1 = 30;
cx2 = cx1 + xw_width * vscale;
cy1 = 30;
cy2 = cy1 + yw_width * vscale;
        prefposition(cx1,cx2,cy1,cy2);
        image_win = winopen("IMAGE WINDOW ");
        ortho2(0,xw_width,0,yw_width);
        RGBmode();
        gconfig();

cx2 = 1200 - 20;
cx1 = cx2 - nn[1] * vscale;
cy2 = cy1 + nn[2] * vscale;

        prefposition(cx1,cx2,cy1,cy2);
        fft1_win = winopen(" FFT number 1");
        ortho2(0,nn[1],0,nn[2]);
        RGBmode();
        gconfig();

cy1 = cy2 + 30;
cy2 = cy1 + nn[2]/2 * vscale;

        prefposition(cx1,cx2,cy1,cy2);
        fft2_win = winopen(" FFT number 2");
        ortho2(0,nn[1],0,nn[2]/2);
        RGBmode();
        gconfig();
#endif


win_rel = 1;
x_offset = 0;

/* compute the number of buffers */
buffer_bytes = xw_width * image_height * 2;

start_buffer = 0;

#ifdef show
printf(" enter the starting buffer : ");
scanf("%d",&start_buffer);
#endif
```

```
end_buffer = n_wind_sec_x;

/*
start_buffer = 30;
end_buffer = 31;
*/

for (buf_loop = start_buffer; buf_loop < end_buffer; buf_loop++)
        {
        get_image_file();
        piv();
        }


/* close the files */
fclose(new_file);
close(image_file);


system(" ps -ef | grep lapointe >> acct.file ");
}


get_image_file()
{
printf(" reading data .\n");
/* Read in all of the data line by line                                    */
/* the data is read into the array " buffer"                               */
/* Since the data went from a 16 bit machine (PC) to a 32 bit machine (SGI) */
/* each element of the array is bit shifted by 8 bits                      */
/* The data on the PC is 1's complement, the data on the SGI is 2's        */
/* complement, so if the data is less than 0 add 256                       */

image_offset_bytes = buf_loop * buffer_bytes;

seek_error = lseek(image_file,image_offset_bytes,SEEK_SET);
if(seek_error == -1)
        {
        printf(" start getting data seek error = : %ld \n ",seek_error);
        exit(1);
        }

for ( j = 0 ; j < xw_width ; ++j)
        {
        n_read = read(image_file,buffer,image_height * 2);
        for( k = 0 ; k < image_height; ++k)
                {
                shade = buffer[k] >> 8;
                if( shade < 0   ) shade += 256;
                if( shade < shade_low   )
                                shade = 0;

                if( shade > shade_high   )
                                shade = shade_high;
```

```
                image_data[j][k] = shade;
                }
        }
}


piv()
{

yw1 = 0;        /*                      y               */
win = 1;        /* first window number                  */

/* in This routine loop through each window, get the two corners of the window  */

        xw1 = 0;
        xw2 = xw_width;
        x_offset = buf_loop * xw_width ;
        xw1out = x_offset;
        xw2out = xw2 + x_offset;

        for (y_win_step = 0 ; y_win_step < n_wind_sec_y; y_win_step++)
                {
                yw1 = y_win_step * yw_width;
                yw2 = yw1 + yw_width;


                fft_piv();

fprintf(new_file,"%d %d %d %d %d %d %d %f %f %f %f \n",
        win_rel,xw1out,yw1,xw2out,yw2,peak_value_x,peak_value_y,
        avg_value_x,avg_value_y,fft_ratio,cen_ratio);


                win++;
                win_rel++;
                }

}

fft_piv()
{
/** Do the FFT on the IMAGE to get the YOUNGS FRINGES */


#ifdef show
show_image();
#endif

pass_one();
two_d_fft();
fft_magnitude();

#ifdef show
```

```
show_mag_1();
#endif


/* Do the FFT on the FRINGES to get the DISPLACEMENT */
pass_two();
two_d_fft();
fft_magnitude();

get_disp();

}

/* ================================================================== */
/* ================================================== */
/* ========================================================= */
pass_one()
{
/* On the first pass */
/* Swap the 2D Real data input array to a 1 D */
/* Complex data input array */
/* zero out the array */
aindex = 1;
        for( k = 0 ; k < nn[2]  ; k++)
                {
                for (j = 0 ; j < nn[1]; j++)
                        {
                        data[aindex] = 0;
                        data[aindex + 1 ] = 0 ;
                        aindex += 2;
                        }
                }
aindex = 1;
        for( k = 0 ; k < yw_width ; k++)
                {
                for (j = 0 ; j < xw_width; j++)
                        {
                        data[aindex] = (float)image_data[j][k + yw1];
                        data[aindex + 1 ] = 0 ;
                        aindex += 2;
                        }
        aindex =  k * nn[1] * 2 + 1;
                }
}
/* ========================================================= */
/* ========================================================= */
/* ========================================================= */
pass_two()
{
/* On the second Pass , Take the Fringe pattern calculated by the */
/* first 2D FFT and do a 2DFFT on the Fringe Pattern */
/* This will give a displacment vector */

        aindex= 1 ;
```

```
        for( k = 0 ; k < nn[2] ; k++)
                {
                for (j = 0 ; j < nn[1]; j++)
                        {
                        data[aindex] = fft_data[j][k];
                        data[aindex + 1 ] = 0 ;
                        aindex += 2;
                        }
                }

}
/* ============================================================ */
/* ============================================================ */
/* ============================================================ */
two_d_fft()
{
float theta_ratio;
int ntot,idim,ndim,n;
int i1,i2,i3;
int ip1,ip2,ip3;
int ibit,i2rev;
int i3rev,i3revi;
int nrem, nprev;
int ifp1,ifp2;

int k1,k2,k3;

float wr,wi,wpr,wpi,wtemp,theta;
float powx, powy;
float tempi,tempr;

/* Set up the sobroutine for a 2D FFT , ndim = 2 */
ndim = 2;
isign = 1;

/* The 2D FFT is taken from the NUMERAICAL RECIPES BOOK */
/* The codewas converted from Fortran to C */

ntot = 1;
for( idim = 1; idim <= ndim; idim++)
        {
        ntot = ntot * nn[idim];
        }

nprev = 1;
for( idim = 1; idim <= ndim; idim++)
        {
        n = nn[idim];
        nrem = ntot/(n * nprev);
        ip1 = 2 * nprev;
        ip2 = ip1 * n;
        ip3 = ip2 * nrem;
        i2rev = 1;
```

```
for( i2 = 1; i2 <= ip2; i2 += ip1)
        {
        if( i2 < i2rev)
                {
                for(i1 = i2; i1 <= i2+ip1 -2; i1 +=2)
                        {
                        for(i3 = i1; i3 <=ip3; i3 +=ip2)
                                {
                                i3rev = i2rev + i3 - i2;
                                tempr = data[i3];
                                tempi = data[i3 + 1];
                                data[i3] = data[i3rev];
                                data[i3 +1] = data[i3rev +1];
                                data[i3rev] = tempr;
                                data[i3rev +1] = tempi;
                                }
                        }
                }
        ibit = ip2/2.0;
        while( (ibit >= ip1) && (i2rev > ibit))
                {
                i2rev = i2rev - ibit;
                ibit = ibit/2.0;
                }
        i2rev = i2rev + ibit;
        }

ifp1 = ip1;

while( ifp1 < ip2)
        {

        ifp2 = 2.0 * ifp1;
        theta = isign * 6.283185 / (ifp2 / ip1);
        wpr = -2.0 * sin(0.50 * theta) * sin(0.50 * theta);
        wpi = sin(theta);

        wr = 1.0;
        wi = 0.0;

        for(i3 = 1; i3 <= ifp1; i3 +=ip1)
                {
                for(i1 = i3; i1 <= i3 + ip1 -2; i1 +=2)
                        {
                        for (i2 = i1; i2 <= ip3; i2 += ifp2)
                                {
                                k1 = i2;
                                k2 = k1 + ifp1;
                                tempr = wr * data[k2] - wi * data[k2 +1];
                                tempi = wr * data[k2 +1] + wi * data[k2];
                                data[k2] = data[k1] - tempr;
                                data[k2+1] = data[k1 +1] - tempi;
                                data[k1] = data[k1] + tempr;
                                data[k1+1] = data[k1+1] + tempi;
```

```
                                        }
                                }
                        wtemp = wr;
                        wr = wr * wpr - wi * wpi + wr;
                        wi = wi * wpr + wtemp * wpi + wi;
                        }

                ifp1 = ifp2;
                }
        nprev = n * nprev;
        }

}
/* ============================================================ */
/* ============================================================ */
/* ============================================================ */
fft_magnitude()
{

/* Compute the Magnitude of the FFT */
aindex= 1 ;
for( k = 0 ; k < nn[2] ; k++)
        {
        for (j = 0 ; j < nn[1]; j++)
                {
                square = data[aindex] * data[aindex] + data[aindex +1 ] * data[aindex +1];
                magnitude = pow(square,0.50);
                fft_data[j][k] =  magnitude ;
                aindex += 2;
                }
        }

}
/* ================================================================== */
/* ========================================================== */
/* ========================================================== */

get_disp()
{

find_peak();
first_peak_value = fft_peak;
first_peak_x = peak_value_x;
first_peak_y = peak_value_y;


#ifdef show
show_mag_2();
#endif

/* zero out the first peak */
j = peak_value_x;
k = peak_value_y;
```

```
aj1 = j - dia_part/2;
if(aj1 < 0) aj1 = 0;
aj2 = j + dia_part/2;
if(aj2 > nn[1]) aj2 = nn[1];

ak1 = k - dia_part/2;
if(ak1 < 0) ak1 = 0;
ak2 = k + dia_part/2;
if(ak2 > nn[1]) ak2 = nn[1];

        for(j = aj1; j <= aj2; j++)
                {
                for(k = ak1; k <= ak2; k++)
                        {
                        fft_data[j][k] = 0;
                        }
                }


/* find the second peak */
find_peak();
second_peak_value = fft_peak;
second_peak_x = peak_value_x;
second_peak_y = peak_value_y;

fft_peak = first_peak_value;
peak_value_x = first_peak_x;
peak_value_y = first_peak_y;


fft_ratio = first_peak_value / second_peak_value;

        if(peak_value_x >= nn[1]/2)
                peak_value_x = peak_value_x - nn[1];

        avg_value_x = peak_value_x;
        avg_value_y = peak_value_y;

}


/* ============================================================ */
/* ============================================================ */
/* ============================================================ */
find_peak()
{
peak_value_x = 0;
peak_value_y = 0;
fft_peak = 0.0;

/* On the Second Pass Of the FFT get the Peak value for the */
/* Displacement Vector . Since the 2D FFT folds over , only look */
/* at half of the map */
        for( k = dia_part  ; k < nn[2]/2 ; k++)
```

```
                {
                for (j = 0 ; j <= dia_part  ; j++)
                        {
                        if(fft_data[j][k] > fft_peak)
                                {
                                fft_peak = fft_data[j][k];
                                peak_value_x = j;
                                peak_value_y = k;
                                }

                        }
                }

        for( k = 0  ; k < nn[2]/2 ; k++)
                {
                for (j = dia_part ; j < nn[1] - dia_part  ; j++)
                        {
                        if(fft_data[j][k] > fft_peak)
                                {
                                fft_peak = fft_data[j][k];
                                peak_value_x = j;
                                peak_value_y = k;
                                }
                        }
                }

        for( k = dia_part  ; k < nn[2]/2; k++)
                {
                for (j = nn[1] - dia_part ; j <= nn[1]  ; j++)
                        {
                        if(fft_data[j][k] > fft_peak)
                                {
                                fft_peak = fft_data[j][k];
                                peak_value_x = j;
                                peak_value_y = k;
                                }

                        }
                }

}

/* ================================================== */
/* ================================================== */
/* ================================================== */
/* ================================================== */
/* ================================================== */
/* ================================================== */

#ifdef show

/* ROUTINES BELOW THIS LINE ARE ONLY USED TO DEBUG THE CODE **/
make_image()
{
```

```
int a1,a2,b1,b2;
int p1x,p1y,p2x,p2y,p3x,p3y;
int pt1;
int pt2;
float ratio_x,ratio_y;
int vel_x,vel_y;
int ix,iy;
int npoints ;

run_number = 1234;
frame_number = 0;
a1 = 0;
a2 = xw_width/2;
b1 = 0;
b2 = yw_width/2 ;

printf(" a %d %d  b %d %d \n",a1,a2,b1,b2);


/* input the maximum expected displacment in the x and y directions */
max_x_disp = 40;
max_y_disp = 40;

ratio_x =    ((float)max_x_disp - 10)/1;
ratio_y =    ((float)max_y_disp - 10)/1;

xwin = image_width;
ywin = image_height;

printf(" enter x vel : " );
scanf("%d",&vel_x);
printf(" enter y vel : ");
scanf("%d",&vel_y);

printf(" number of points: ");
scanf("%d",&npoints);


for( k = 0 ; k < npoints; ++k)
                {
                pt1 = (float)rand()/100;
                pt2 = (float)rand()/100;
                while (pt1 < a1 || pt1 > a2 || pt2 < b1  || pt2 > b2)
                        {
                        pt1 = (float)rand()/100;
                        pt2 = (float)rand()/100;
                        }
                        p1x = pt1 + j * xw_width;
                        p1y = pt2 + i * yw_width;
                        p2x = p1x + vel_x ;
                        p2y = p1y + vel_y ;
                        p3x = p1x + vel_x *2  ;
                        p3y = p1y + vel_y *2  ;
```

```
                        l = p1x;
                        m = p1y;
                        make_dot();
                        l = p2x;
                        m = p2y;
                        make_dot();
                        l = p3x;
                        m = p3y;
                        make_dot();

                }



}

make_dot()
{
                        image_data[l][m] = 255;

/*
                        image_data[l-1][m] = 200;
                        image_data[l+1][m] = 200;
                        image_data[l][m+1] = 200;
                        image_data[l][m-1] = 200;
*/
}

show_image()
{
        skip = 1;
        winset(image_win);
        rgbvector[0] = 200;
        rgbvector[1] = 200;
        rgbvector[2] = 0;

printf(" in show image x is %d %d y is %d %d \n",xw1,xw2,yw1,yw2);

                for ( j = 0 ; j < xw_width ; j+= skip  )
                {
                        for( k = 0 ; k < yw_width; k+= skip)
                                {
                                rgbvector[0] = image_data[j + xw1][k + yw1];
                                rgbvector[1] = image_data[j + xw1][k + yw1];
                                rgbvector[2] = image_data[j + xw1][k + yw1];

                                c3s(rgbvector);
                                rectfi(j,k,j+1,k+1);
                                }
                }

}
```

```
show_mag_1()
{
float max_image;
float min_image;
float slope, b;

min_image = 1000;
max_image = 0;


winset(fft1_win);
        rgbvector[0] = 200;
        rgbvector[1] = rgbvector[2] = 0;
        c3s(rgbvector);
        clear();

                for ( j = 0 ; j < nn[1] ; j++ )
                        {
                        for( k = 0 ; k < nn[2]; k++)
                                {
                                if( fft_data[j][k] > max_image)
                                        {
                                        max_image = fft_data[j][k];

                                        }
                                if( fft_data[j][k] < min_image)
                                        {
                                        min_image = fft_data[j][k];

                                        }
                                }
                        }


slope = 255.0 / (max_image - min_image);
b = slope * min_image;


                for ( j = 0 ; j < nn[1]; j++ )
                        {
                        for( k = 0 ; k < nn[2]; k++)
                                {

cp = fft_data[j][k] * slope - b ;


/*
get_color();
*/
        rgbvector[0] = cp ;
        rgbvector[1] = cp ;
        rgbvector[2] = cp ;
c3s(rgbvector);
rectfi(j,k,j+1,k+1);
```

```
/*
pnt2i(j,k);
*/                              }
                        }


}


show_mag_2()
{
float max_image;
float min_image;
float slope, b;

min_image = 1000;
max_image = 0;

winset(fft2_win);


        rgbvector[0] = 200;
        rgbvector[1] = rgbvector[2] = 0;
        c3s(rgbvector);
        clear();

                for ( j = 0 ; j < dia_part ; j++ )
                        {
                        for( k = dia_part ; k < nn[2]/2; k++)
                                {
                                if( fft_data[j][k] > max_image)
                                        {
                                        max_image = fft_data[j][k];

                                        }
                                if( fft_data[j][k] < min_image)
                                        {
                                        min_image = fft_data[j][k];

                                        }
                                }
                        }
                for ( j = dia_part ; j < nn[1] - dia_part ; j++ )
                        {
                        for( k = 0 ; k < nn[2]/2; k++)
                                {
                                if( fft_data[j][k] > max_image)
                                        {
                                        max_image = fft_data[j][k];

                                        }
                                if( fft_data[j][k] < min_image)
                                        {
```

```
                                        min_image = fft_data[j][k];

                                        }
                                }
                        }
/*
printf(" max image = %f min = %f \n",max_image, min_image);

max_image = 2000000.0;
min_image = 10000.0;
*/

slope = 255.0 / (max_image - min_image);
b = slope * min_image;


                for ( j = 0 ; j < nn[1] ; j++ )
                        {
                        for( k = 0 ; k < nn[2]; k++)
                                {

cp = fft_data[j][k] * slope - b;

get_color();
c3s(rgbvector);


rectfi(j,k,j+1,k+1);

/*
                                        pnt2i(j,k);
*/
                                        }
                        }


}

get_color()
{

        rgbvector[1] = cp ;
        rgbvector[1] = 0;
        rgbvector[2] = 0;

if(cp < 250)
        {
        rgbvector[0] = 0;
        rgbvector[1] = cp * 255.0/250.0 ;
        rgbvector[2] = 0;
        }
if(cp < 245)
        {
        rgbvector[0] = 0;
```

```
        rgbvector[1] = 0;
        rgbvector[2] = cp * 255.0/245.0;
        }
if(cp < 240)
        {
        rgbvector[0] = 0;
        rgbvector[1] = cp * 255.0/240.0;
        rgbvector[0] = cp * 255.0/240.0 ;
        }
if(cp < 230)
        {
        rgbvector[0] = cp * 255.0/230.0;
        rgbvector[1] = 0;
        rgbvector[2] = cp * 255.0/230.0 ;
        }
if(cp < 220)
        {
        rgbvector[0] = cp ;
        rgbvector[1] = cp ;
        rgbvector[2] = cp ;
        }
if(cp < 10)
        {
        rgbvector[0] = 0 ;
        rgbvector[1] = 0 ;
        rgbvector[2] = 0 ;
        }


}

#endif
```

Appendix H

2D FFT PIV CODE FOR THE CRAY: CRAY_FFTPIV.C

```
/**    PARTICLE IMAGE VELOCEMETRY PROGRAM     **/
/**      Using the 2D fft method                **/
/**    By :   Kenneth LaPointe          **/
/**          NUSC, Code 8322           **/
/**    Last rev as of  Aug 17 1992       **/
/** This program reads in data files and image  **/
/** files to use an 2D FFT program          **/
/** to compute the displacemnet between     **/
/** particles                       **/
/** compile on CRAY ONLY */

/** Run this Script to compile the code on the cray
   and move it to the working directory

echo Copy the Source code to the CRAY
rcp fftpiv.c cray:fftpiv.c

echo  compile The source code
rsh cray cc -O3 -c fftpiv.c

echo  Link the code
rsh cray segldr -lm fftpiv.o -o fftpiv

echo Make the tmp user directory
rsh cray mkdir /usr/tmp/PIV

echo move the executable code to the tmp dir
rsh cray mv fftpiv /usr/tmp/PIV

********************* */

#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>

#include <complex.h>

int end_buffer;
int fft_loop ;
int fft_pass;
int i,j,k,l,m,n,o,p;
int aj1,aj2,aj3,ak1,ak2,ak3;
float a,b,c,d,e,f,g;

double square, magnitude;

float fft_peak;
float fft_ratio;
float cen_ratio;


float first_peak_value;
int first_peak_x;
```

```
int first_peak_y;

float second_peak_value;
int second_peak_x;
int second_peak_y;
int aindex;

int junk;
int nbuff;
int buffer_bytes;
int buf_loop,x_offset,start_buffer;
int pic_width;
int win_rel;

short rgbvector[3];
long datastore;

int xw1,yw1;   /* lower left corner of 2 D array is xw1,yw1        */
int xw2,yw2;   /* upper right corner of 2 D array is xw2,yw2       */

int xw_width,yw_width;

int n_windows,win;
int max_frames,run_number;
int x_win_step,y_win_step;
int total;

int xw1out,xw2out;
int max_x_disp,max_y_disp;

int image_file;
int image_height,image_width;

char old_disp_name[50];
char new_disp_name[50];

int shade;
int n_wind_sec_x, n_wind_sec_y;
int frame_number;
int dia_part;
int xwin,ywin;
float skipx, skipy;


int draw_data;
int shift;

int  peak_value_x;
int  peak_value_y;
float avg_value_x;
float avg_value_y;

float x_sum,y_sum,x_weight,y_weight;
float x_sum_total,y_sum_total;
```

```
float zero_limit;

int shade_low,shade_high;
long buffer[1024];
unsigned long s1,s2,s3,s4;
int array_index = 300;
complex double complex_data[300][300];
float fft_data[300][300];
float table[2048];
int ntable;

off_t seek_error,image_offset_bytes;
unsigned short image_data[256][3200];
int max_array_width = 256;
int max_array_height = 3000;

int win_buf;

int  skip;
int buf_start;

int fd,n_read,n_write,bytes;          /* file descriptors */
FILE *new_file,*old_file,*fopen();

int isign;
int n_fft_x,n_fft_y;


main(argc,argv)
int argc;
char *argv[];
{
printf(" Start of Program \n");

/* ====================== REQUIRED INPUTS ============================        */
/* A 35 mm negitive  is 36mm x 26 mm , with the Nikon scanner at a pitch of 1 */
/* the negative is digitized to 125 pixels/mm or 4500 x 3000 pixels          */
/* For use on the Cray the size of the buffer and picture  must be an integer        */
/* multiple of 4 */

image_height = 3000;
image_width = 4500;
shade_low = 10;
shade_high = 240;

/* The program must know the diameter of a particle so that the */
/* zero  position is skipped */
dia_part = 6;

/* ============================================================ */
/* compute the number of windows to break the image into. This will be the number */
/* of displacecent vectors per image ,  set the number so that there are         */
/* about 8 particle per window */
```

```
xw_width = yw_width = 40;

/* The number of points in the fft value should be either                */
/*        16,32,64,128,or 256                                            */
/* the value must be greater than the window width                       */
n_fft_x = 128;
n_fft_y = 128;

/*
printf(" Check memory allocation \n");

if( n_fft_x > array_index || n_fft_y > array_index)
        {
        printf(" not enough memory in fft arrays \n");
        exit(1);
        }


if(image_height > max_array_height )
        {
        printf(" height too big \n");
        exit(1);
        }

if(xw_width > max_array_width )
        {
        printf(" xw_width too big \n");
        exit(1);
        }

*/


/* for the each input buffer */
n_wind_sec_y = image_height / yw_width;
n_wind_sec_x = image_width  / xw_width;      /* number of windows in the x direction */
n_windows = n_wind_sec_x * n_wind_sec_y;    /* total number of windows and vectors  */


printf(" Open Data Files \n");
/*** open the Image data file ***/
image_file = open(argv[1],0);

/*** open the displacment data file **/
new_file = fopen(argv[2],"w");

fprintf(new_file,"%d \n",run_number);
fprintf(new_file,"%d \n",frame_number);
fprintf(new_file,"%d \n",image_width);
fprintf(new_file,"%d \n",image_height);
fprintf(new_file,"%d \n",xw_width);
fprintf(new_file,"%d \n",yw_width);
fprintf(new_file,"%d \n",n_wind_sec_x);
fprintf(new_file,"%d \n",n_wind_sec_y);
```

```
fprintf(new_file,"%d \n",n_fft_x);
fprintf(new_file,"%d \n",n_fft_y);

win_rel = 1;
x_offset = 0;

/* compute the number of buffers */
buffer_bytes = xw_width * image_height * 2;


printf(" Start The x loop \n");
start_buffer = 0;
end_buffer = n_wind_sec_x;

for (buf_loop = start_buffer; buf_loop < end_buffer; buf_loop++)
        {
        get_image_file();
        piv();
        }


/* close the files */
fclose(new_file);
close(image_file);


system(" ps -ef | grep lapointe >> acct.file ");
}


get_image_file()
{
        /* Read in all of the data line by line                    */
        /* the data is read into the array " buffer"                */
                /* Since the data went from a 16 bit machine (PC) to a 64 bit machine (CRAY) */
        /* 1 word on the cray contains 4 PC words                  */

        image_offset_bytes = buf_loop * buffer_bytes;
        seek_error = lseek(image_file,image_offset_bytes,SEEK_SET);

for ( j = 0 ; j < xw_width ; ++j)
        {
        m = 0;
        n_read = read(image_file,buffer,image_height * 2);
        for( k = 0 ; k < image_height/4; k++)
                {
                image_data[j][m] = image_data[j][m+1] = image_data[j][m+2] =
        image_data[j][m+3] = 0;
                s1 = buffer[k] ;
                image_data[j][m ] = s1 >> 56;
                if(image_data[j][m] < shade_low || image_data[j][m] > 255)
image_data[j][m] = 0;

                s2 = buffer[k] << 16;
```

```
                image_data[j][m+1] = s2 >> 56;
                if(image_data[j][m+1] < shade_low || image_data[j][m+1] > 255)
        image_data[j][m+1] = 0;

                s3 = buffer[k] << 32;
                image_data[j][m+2] = s3 >> 56;
                if(image_data[j][m+2] < shade_low || image_data[j][m+2] > 255)
        image_data[j][m+2] = 0;

                s4 = buffer[k] << 48;
                image_data[j][m+3] = s4 >> 56;
                if(image_data[j][m+3] < shade_low || image_data[j][m+3] > 255)
        image_data[j][m+3] = 0;


                m +=4;
                }
        }
}


piv()
{

        yw1 = 0;
        win = 1;
        xw1 = 0;
        xw2 = xw_width;
        x_offset = buf_loop * xw_width ;
        xw1out = x_offset;
        xw2out = xw2 + x_offset;

        for (y_win_step = 0 ; y_win_step < n_wind_sec_y; y_win_step++)
                {
                yw1 = y_win_step * yw_width;
                yw2 = yw1 + yw_width;

                fft_piv();

                fprintf(new_file,"%d %d %d %d %d %d %d %f %f %f %f \n",
                        win_rel,xw1out,yw1,xw2out,yw2,peak_value_x,peak_value_y,
                        avg_value_x,avg_value_y,fft_ratio,cen_ratio);
                win++;
                win_rel++;
                }
}

fft_piv()
{
        /** Do the FFT on the IMAGE to get the YOUNGS FRINGES */
        pass_one();
        two_d_fft();
        fft_magnitude();
```

```
        /* Do the FFT on the FRINGES to get the DISPLACEMENT */
        pass_two();
        two_d_fft();
        fft_magnitude();

        get_disp();

}

/* ============================================================ */
/* ============================================================ */
/* ============================================================ */
pass_one()
{
/* On the first pass set the array to zero */
        for( k = 0 ; k < n_fft_y  ; k++)
                {
                for (j = 0 ; j < n_fft_x; j++)
                        {
                        complex_data[j][k] = 0 + 0i;
                        }
                }

/* Then input the data into the complex array */
        for( k = 0 ; k < yw_width ; k++)
                {
                for (j = 0 ; j < xw_width; j++)
                        {

                        complex_data[k][j] = (float)image_data[j][k + yw1] + 0i;
                        }
                }


}
/* ============================================================ */
/* ============================================================ */
/* ============================================================ */
pass_two()
{
/* On the second Pass , Take the Fringe pattern calculated by the */
/* first 2D FFT and put it into the input array for the next pass*/


        for( k = 0 ; k < n_fft_y ; k++)
                {
                for (j = 0 ; j < n_fft_x; j++)
                        {

                        complex_data[k][j] = fft_data[j][k] + 0i;
                        }
                }

}
```

```
/* ========================================================== */
/* ========================================================== */
/* ========================================================== */
two_d_fft()
{
        float work[263000];
        float scale;

        int isign,n1,n2;
        int incr1;
        int nwork;
        int incr;

        isign = 1;
        n1 = n_fft_x;
        n2 = n_fft_y;
        scale = 1.0;
        incr = 1;
        skip = n_fft_x;

        ntable = 2048;

        nwork = 262144;

        /* See the CRAY C manaual for the discussion of this subroutine */
        CFFT2D(&isign,&n1,&n2,&scale,complex_data,&incr,&array_index,
                complex_data,&incr,&array_index,table,&ntable,work,&nwork);
}

/* ========================================================== */
/* ========================================================== */
/* ========================================================== */
fft_magnitude()
{
        /* Compute the Magnitude of the FFT */
        for( k = 0 ; k < n_fft_y ; k++)
                {
                for (j = 0 ; j < n_fft_x; j++)
                        {
                        magnitude = cabs(complex_data[k][j]);
                        fft_data[j][k] =  magnitude ;
                        }
                }

}

/* ========================================================== */
/* ========================================================== */
/* ========================================================== */

get_disp()
{
/* find highest peak in the 2D FFT data */
find_peak();
```

```
first_peak_value = fft_peak;
first_peak_x = peak_value_x;
first_peak_y = peak_value_y;


/* zero out the first peak data to find the next highest peak */
j = peak_value_x;
k = peak_value_y;

aj1 = j - dia_part/2;
if(aj1 < 0) aj1 = 0;
aj2 = j + dia_part/2;
if(aj2 > n_fft_x) aj2 = n_fft_x;

ak1 = k - dia_part/2;
if(ak1 < 0) ak1 = 0;
ak2 = k + dia_part/2;
if(ak2 > n_fft_y) ak2 = n_fft_y;

        for(j = aj1; j <= aj2; j++)
                {
                for(k = ak1; k <= ak2; k++)
                        {
                        fft_data[j][k] = 0;
                        }
                }


/* find the second highest peak in the 2D FFT data*/
find_peak();
second_peak_value = fft_peak;
second_peak_x = peak_value_x;
second_peak_y = peak_value_y;


fft_peak = first_peak_value;
peak_value_x = first_peak_x;
peak_value_y = first_peak_y;


        fft_ratio = first_peak_value / second_peak_value;

/* Decide weather the peak is in the first or second quadrant */
        if(peak_value_x >= n_fft_x / 2)
                peak_value_x = peak_value_x - n_fft_x;

/* For now the avg_peak is the same as the peak value */
/* Later code will be added to compute the average displacement around the peak */
        avg_value_x = peak_value_x;
        avg_value_y = peak_value_y;


}
```

```
/* ================================================== */
/* ================================================== */
/* ================================================== */
find_peak()
{
peak_value_x = 0;
peak_value_y = 0;
fft_peak = 0.001;

/* After the Second Pass of the 2D FFT get the Peak value for the */
/* Displacement Vector . Since the 2D FFT folds over , only look */
/* at half of the map */

        for( k = 0  ; k < dia_part ; k++)
                {
                for (j = dia_part ; j < n_fft_x -  dia_part -1  ; j++)
                        {
                        if(fft_data[j][k] > fft_peak)
                                {
                                fft_peak = fft_data[j][k];
                                peak_value_x = j;
                                peak_value_y = k;
                                }

                        }
                }


        for( k = dia_part ; k < n_fft_y/2 -1; k++)
                {
                for (j = 0; j < n_fft_x -1 ; j++)
                        {
                        if(fft_data[j][k] > fft_peak)
                                {
                                fft_peak = fft_data[j][k];
                                peak_value_x = j;
                                peak_value_y = k;
                                }
                        }
                }

}
```

Appendix I

VECTOR DISPLAY CODE: VECTOR.C

```
/**       PARTICLE IMAGE VELOCEMETRY DISPLAY PROGRAM   **/
/** Draw vectors on screen and in file for  hp plotter    **/
/**       By :      Kenneth LaPointe                 **/
/**                 NUSC, Code 8322                    **/
/**       REV C. , Last rev as of 20 Feb 1992     **/
/**      to compile  type : cc vector.c -lgl_s -lm -o vector **/
#include <gl/gl.h>

#include <stdio.h>
#include <math.h>

int true = 1;
int false = 0;
int window_too_small;
int i,j,k,l,m,n,o,p;
int junk;
int win_width_x;
int win_width_y;
int xw_width,yw_width;
int n_windows,window_number;
int frame, max_frames,run_number,frame_step;
int x_win_step,y_win_step,win_steps;
int total,max_corr;
int old_x_peak, old_y_peak;
int xd_peak,yd_peak;

int disp_shift;

float avg_xd,avg_yd;
float x_end,y_end,x_center,y_center;
float magnitude;
float angle , yx, h_angle;
float xp1,xp2,xp3,xp4,yp1,yp2,yp3,yp4;
int winx,winy;

float a,b,c,d,e,f;

float r1,r2,r3,r4,r5,r6,r7,r8;
float rl1,rh1;
int da1,da2,da3,da4;

float window_x,window_y;
float fft_ratio;
float grid_ratio;

char file_name[30];              /* file name character storage */
char title[30];
char last_data_file[20];

int image_height,image_width;
int xoffset;


char input_name[100];
```

```
int shade;
int n_wind_sec_x, n_wind_sec_y;
int frame_number;
int disp_file;
int lstep;
int xw1,xw2,yw1,yw2;
int fd,n_read,n_write,bytes;            /* file descriptors */
float win_scale;
int cmap;
int red,green,blue;
int wn;
int x_peak,y_peak;
float vector [15000][6]; /*     0 magnitude
                                1 angle
                                2 x center
                                3 y center
                                4 fft_ratio
                                5 quality       */

int xfft,yfft;
float max_disp;
int ix;
float scale_factor;
float dim_ratio;
float laser_freq ;
int xmar,ymar;

float max_mag, mratio,ncolors;
int pens,pratio;
int pen_number;
int xa,xb;

int showflag , closeflag,ratioflag;
 int part_dia;

float m0,a0,mv,av,lowl,highl,error_limit;
float ratio_limit;

FILE *sfile,*dsp_file,*old_file,*fp,*file_in,*hp,*fopen();

/* ************************************************************** */


main(argc,argv)
int argc;
char *argv[];
{

foreground();

file_in = fopen("holder.dat","w");
fprintf(file_in,"%s",argv[1]);
fclose(file_in);
file_in = fopen("holder.dat","r");
```

```
fscanf(file_in,"%s",file_name);
fclose(file_in);

disp_shift = 0;
cmap = 1000;
ncolors = 10;
pens = 6;
scale_factor = 4;
h_angle = 5. * (2. * 3.14 / 360.0);
max_disp = 100.0;



printf(" enter the ratio limit : ( 0 shows all vectors) ");
scanf("%f",&ratio_limit);

part_dia = -1;
if( ratio_limit != 0)
{
        printf(" enter the particle diameter (pixels) : ");
        scanf("%d",&part_dia);
}

/* set up the limit for neighboring vectors */
error_limit = 0.25;
lowl = 1.0 - error_limit;
highl = 1.0 + error_limit;


/* set the ratio to get from pixels to ft/sec */
dim_ratio = 0.0520 ;     /* ft/sec per pixel */


printf(" OPEN DATA FILE HEADER  %s \n",file_name);
dsp_file = fopen(file_name,"r");
read_datafile_header();
setup_plotter();


printf(" Read in the DATA FILE \n");

xw1 = 0;
yw1 = 0;
xoffset = 0;
        for(wn = 0 ; wn < n_windows; wn++)
                {
                get_disp();
                }
fclose(dsp_file);


/* Set up the color scale */
mratio = ncolors / (max_mag * scale_factor);
pratio = pens / (max_mag * scale_factor);
```

```
printf(" colors %f , max %f  ratio  %f \n",ncolors,max_mag,mratio);


select_data();
interpolate();

open_vector_window();

if(ratio_limit == 0)
        {
        for(wn = 0 ; wn < n_windows; wn++)
                {
                        color(1);
                        compute_vector();
                        screen_arrow();
                        paper_arrow();
                }

        }
else
        {
        for(wn = 0; wn < n_windows; wn++)
                {
                        if(vector[wn][5] ==1)
                                {
                                color(0);
                                compute_vector();
                                screen_arrow();
                                paper_arrow();
                                }

                        if(vector[wn][5] == 2)
                                {
                                color(1);
                                compute_vector();
                                screen_arrow();
                                paper_arrow();
                                }

                }
        }

fclose(hp);


printf(" finished dumping plot data \n");

sleep(960);

}



read_datafile_header()
```

```
{

        fscanf(dsp_file,"%d ",&run_number);
        fscanf(dsp_file,"%d ",&frame_number);
        fscanf(dsp_file,"%d ",&image_width);
        fscanf(dsp_file,"%d ",&image_height);
        fscanf(dsp_file,"%d ",&xw_width);
        fscanf(dsp_file,"%d ",&yw_width);
        fscanf(dsp_file,"%d ",&n_wind_sec_x);
        fscanf(dsp_file,"%d ",&n_wind_sec_y);
        fscanf(dsp_file,"%d ",&xfft);
        fscanf(dsp_file,"%d ",&yfft);

        n_windows = n_wind_sec_x * n_wind_sec_y;
        ix = n_wind_sec_y;

/*
        printf(" %d \n",run_number);
        printf(" frame number %d \n",frame_number);
        printf(" image width  %d \n",image_width);
        printf(" image height %d \n",image_height);
        printf(" number of window secs in x %d \n",n_wind_sec_x);
        printf("                    in y %d \n",n_wind_sec_y);
*/
}


setup_plotter()
{

/* Open the HP plotter file */
printf(" open the Plotter file and set it up \n");
        hp = fopen("vector.hp","w");
        fprintf(hp,"SP1; \n");
        fprintf(hp,"PT.3; \n");
        fprintf(hp,"DT\03; \n");


winx = image_width * 11/8.5;
winy = image_height;

lstep = 0.02 * winx;

a = 1.0/125.0;

        fprintf(hp,"SC -100,%d,-100,%d; \n",winx,winy);
/* Draw Run ID info */
        xmar = winx;
        ymar = image_height - 100;

fprintf(hp," PU%d,%d;DI0,-1;LBFILE NAME : %s \03 \n",
                xmar  ,ymar,file_name);

fprintf(hp," PU%d,%d;DI0,-1;LBRUN ID NUMBER : %d \03 \n",
```

```
                xmar - lstep,ymar,run_number);

fprintf(hp," PU%d,%d;DI0,-1;LBFrame Number  : %d \03 \n",
                xmar - 2 * lstep,ymar,frame_number);

fprintf(hp," PU%d,%d;DI0,-1;LBPicture size  : %d  X   %d pixels \03 \n",
                xmar - 3 * lstep,ymar,image_width,image_height);

fprintf(hp," PU%d,%d;DI0,-1;LBWindow  size  : %d  X   %d pixels \03 \n",
                xmar - 4 * lstep,ymar,xw_width,yw_width);

l = image_width / xw_width;
m = image_height / yw_width;
n = l * m ;

fprintf(hp," PU%d,%d;DI0,-1;LBVector Count  : %d  X   %d   : total %d \03 \n",
                xmar - 5 * lstep,ymar,l,m,n);

fprintf(hp," PU%d,%d;DI0,-1;LBConversion    : 1 pixel = %f ft/sec \03 \n",
                xmar - 6 * lstep,ymar,dim_ratio);

        fprintf(hp," PU%d,%d;DI1,0 ;LB +X axis \03 \n",image_width + lstep,lstep);
        fprintf(hp," PU%d,%d;DI0,-1;LB +Y axis \03 \n",-lstep,image_height - lstep);

/* draw box */

        fprintf(hp," PU%d,%d; \n",0,0);
        fprintf(hp," PD%d,%d; \n",image_width + 5 * lstep,0);
        fprintf(hp," PU%d,%d; \n",0,0);
        fprintf(hp," PD%d,%d; \n",0,image_height + 5 * lstep);
        fprintf(hp," PU%d,%d; \n",image_width,0);
        fprintf(hp," PD%d,%d; \n",image_width,image_height);
        fprintf(hp," PD%d,%d; \n",0,image_height);

        fprintf(hp," PU%d,%d;DI,0,-1;LBScale: 50 pixels  = %f ft/sec \03 \n",
        image_width + lstep,image_height - 4 * lstep, 50.0 * dim_ratio);

        fprintf(hp,"PT.1; \n");

        xp1 = image_width + lstep;
        yp1 = image_height -  2 * lstep;

        wn = n_windows + 1;

        /* make the vector scale key */
        vector[wn][0] = 50 * scale_factor;
        vector[wn][1] = 0;
        vector[wn][2] = image_width + lstep/2.;
        vector[wn][3] = image_height - 2 * lstep;
        compute_vector();
        paper_arrow();

        vector[wn][1] = 90 * 2 * 3.14159 / 360;
        compute_vector();
```

```
        paper_arrow();

}

open_vector_window()
{
        printf(" FLOW VISUALIZATION PROJECT , DISPLAY PROGRAM \n");
        prefposition(10,4500/3.5 ,10,3000/3.5);
        winopen(" TEST IMAGE ");
        make_cmap();
        ortho2(0,4500,0,3000);
        color(7);
        clear();
        color(3);
}

get_disp()
{

        fscanf(dsp_file,"%d %d %d %d %d %d %d %f %f %f %f \n ",&window_number,

&xw1,&yw1,&xw2,&yw2,&x_peak,&y_peak,&avg_xd,&avg_yd,&fft_ratio,&grid_ratio);

        /* use the peak values */
        avg_xd = x_peak;
        avg_yd = y_peak;

/* get rid of the particle size limited magnitudes */

        if(avg_xd == part_dia || avg_yd == part_dia)
                {
                avg_xd = 0;
                avg_yd = 0;
                }


        magnitude = pow(avg_xd * avg_xd + avg_yd * avg_yd,.5);

        /* Assume a +x and +y flow */
        if(avg_xd == 0 )
                {
                angle = 1.57; /* 90 deg up */
                }
                else
                {
                yx = avg_yd/avg_xd;
                angle = fatan(yx);
                }

        if( angle <= 0)
                angle = angle + 3.14159;
```

```
        vector[wn][0] = magnitude * scale_factor;
        vector[wn][1] = angle;
        vector[wn][2] = (xw1 + xw2) /2.0;
        vector[wn][3] = (yw1 + yw2) /2.0;
        vector[wn][4] = fft_ratio;
        vector[wn][5] = 0;


/* get the peak magnitude for the color scale */
        if (magnitude > max_mag)
                max_mag = magnitude;

}



select_data()
{
for (wn = 0; wn < n_windows; wn++)
        {
        showflag = closeflag = ratioflag = 0;

        /* test for the peak to peak ratio */
        if(vector[wn][4] >= ratio_limit )
                ratioflag = 100;

        /* Test for a vectors with the close to same magnitude and angle */
        m0 = vector[wn][0];
        a0 = vector[wn][1];

 /** Loop through the 3 x 3 matrix of vectors centered around vector[wn] */
        for(j = -1 ; j <= 1 ; j++)
                {
                for(k = -ix ; k <= ix ; k += ix)
                        {
                        if( j + k != 0)
                                {
                                mv = vector[wn + j + k][0];
                                av = vector[wn + j + k][1];

                                if( ((mv <= m0 * highl) && (mv >= m0 * lowl)) &&
                                   ((av <= a0 * highl) && (av >= a0 * lowl))      )
                                        {
                                        closeflag += 3;
                                        }
                                }
                        }
                }

        showflag = ratioflag + closeflag;

        /* greater than confidence ratio and more than 1 close neighbors */
        if(showflag > 103)
                {
```

```
                vector[wn][5] = 1;
                }

        }


}


interpolate()
{
float sum_mag;
float sum_ang;
float sum_vec;

/* Interpolate where there is no data */
for(wn = 0  ; wn < n_windows; wn++)
        {
        sum_mag = sum_ang = sum_vec = 0;
        if(vector[wn][5] == 0)
                {
                /** Loop through the 3 x 3 matrix of vectors centered around vector[wn] */
                for(j = -1 ; j <= 1 ; j++)
                        {
                        for(k = -ix ; k <= ix ; k += ix)
                                {
                                if( j + k != 0)
                                        {
                                        if(vector[wn + j + k][5] == 1 )
                                                {
/*
printf(" %d %d %f %f     %f %f \n",wn,wn + j + k,
vector[wn][0],vector[wn + j + k ][0],vector[wn][1],vector[wn + j + k ][1]);
*/
                                                sum_mag += vector[wn + j + k][0];
                                                sum_ang  = vector[wn + j + k][1];
                                                sum_vec++;
                                                }
                                        }
                                }

                        }
                        if(sum_vec >= 2)
                        {
                        vector[wn][0] = sum_mag / sum_vec;
                        vector[wn][1] = sum_ang ;
                        vector[wn][5] = 2;
                        }
                }
        }
}

compute_vector()
{
```

```
                magnitude = vector[wn][0];
                angle     = vector[wn][1];

                xp1 = vector[wn][2];
                yp1 = vector[wn][3];
                xp2 = vector[wn][2] + magnitude * cos(angle);
                yp2 = vector[wn][3] + magnitude * sin(angle);

                xp3 = xp1 + (magnitude * 0.75) * cos(angle + h_angle);
                yp3 = yp1 + (magnitude * 0.75) * sin(angle + h_angle);

                xp4 = xp1 + (magnitude * 0.75) * cos(angle - h_angle);
                yp4 = yp1 + (magnitude * 0.75) * sin(angle - h_angle);

}


screen_arrow()
{
/*
                i = cmap + magnitude * mratio;
                color(i);
*/
                /* draw arrow  body */
                move2i(xp1,yp1);
                draw2i(xp2,yp2);
                /* draw arrow head  */
/*
                draw2i(xp3,yp3);
                move2i(xp2,yp2);
                draw2i(xp4,yp4);
*/

}




paper_arrow()
{

                /*
                pen_number = pratio * vector[wn][0];
                fprintf(hp,"SP%d ;\n"),pen_number);
                */

                /* draw arrow  */
                fprintf(hp," PU%f,%f; \n",xp1,yp1);
                fprintf(hp," PD%f,%f; \n",xp2,yp2);
                /* draw arrow head */
/*
                fprintf(hp," PD%f,%f; \n",xp3,yp3);
                fprintf(hp," PU%f,%f; \n",xp2,yp2);
```

```
                fprintf(hp," PD%f,%f; \n",xp4,yp4);
                fprintf(hp," \n");
*/


}

make_cmap()
{

/****** make color map ***/
        k = 0;
        l = 256  * 6 / ncolors;
/* black to red */
        red = 0;
        green = 0;
        blue = 0;
        for( j = 0; j < 255 ; j = j + l)
                {
                red = j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }
/* red to purple */
        red = 255;
        green = 0;
        blue = 0;
        for( j = 0; j < 255 ; j = j + l)
                {
                blue = j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }
/* purple to blue */
        red = 255;
        green = 0;
        blue = 255;
        for( j = 0; j < 255 ; j = j + l)
                {
                red = 255 - j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }
/* blue to aqua */
        red = 0;
        green = 0;
        blue = 255;
        for( j = 0; j < 255 ; j = j + l)
                {
                green = j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }
```

```
/* aqua to green */
        red = 0;
        green = 255;
        blue = 255;
        for( j = 0; j < 255 ; j = j + 1)
                {
                blue  = 255 - j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }

/* green to yellow */
        red = 0;
        green = 255;
        blue = 0;
        for( j = 0; j < 255 ; j = j + 1)
                {
                red = j;
                mapcolor(cmap + k,red,green,blue);
                k++;
                }

printf(" Cmap is %d and cmap + k is %d \n",cmap,cmap + k);
}
```

DISTRIBUTION LIST

Internal

Codes: 102 (K. Lima)
0251
0261
0262 (2)
02244
22
38
3891
81
8214 (J. Castano)
83
8321 (W. Barker, B. Sjoblom)
8322 (P. Lefebvre)
8323 (K. LaPointe (3))

Total: 18